Aalto University
School of Science
Degree Programme in Computer, Communication and Information Sciences

Sergei Chistiakov

Secure storage and transfer of data in a smart lock system

Master's Thesis
Espoo, March 17, 2017

Supervisor: Professor Pekka Orponen
Advisors: Professor Kaisa Nyberg, Aalto University
Jarkko Juntunen M.Sc. (Tech.), Lukoton Experience Oy

Aalto University
School of Science
Degree Programme in Computer, Communication and Information Sciences

ABSTRACT OF MASTER'S THESIS

Author:	Sergei Chistiakov		
Title:	Secure storage and transfer of data in a smart lock system		
Date:	March 17, 2017	**Pages:**	72
Major:	Computer Science	**Code:**	SCI3042
Supervisor:	Professor Pekka Orponen		
Advisors:	Professor Kaisa Nyberg Jarkko Juntunen M.Sc. (Tech.)		

The Internet of Things plays a bigger and bigger role in our everyday life. One example of IoT devices are smart locks. Lukoton Experience Oy is a Finnish company working on smart lock systems for businesses. The aims of this work were to analyse their existing smart lock system from the security perspective and to design a new system using the opportunities of an Atmel Corporation EEPROM chip with embedded cryptographic functions. Subsequent to this, the new design was compared to the existing system as well as to similar products on the market. The new system uses AES encryption in CCM mode, stores the keys only in a protected EEPROM chip and the cloud server, and uses Bluetooth Low Energy and HTTPS channels for sending data. The paper presents the new system in detail and shows that it is more secure than the existing one by providing authentication and message integrity as well as better protection of the chip at hardware and software level. It rejects all common attacks and stops Replay attacks at an earlier stage. The new system also resists attacks that many analogues on the market are susceptible to.

Keywords:	AES, Authenticity, Bluetooth Low Energy, Cryptography, Data Exchange, EEPROM Chip, Encryption, Information Security, Internet of Things, Man-in-the-Middle Attack, Message Integrity, Passive Eavesdropping Attack, Replay Attack, Smart Locks
Language:	English

Acknowledgements

This chapter is dedicated to expressing my gratitude to everyone who helped me with this Master's thesis and during my times at Aalto University.

First of all, I would like to thank my thesis advisor Professor Kaisa Nyberg. I cannot overestimate how much she helped me with progressing on this thesis. Kaisa was my professor at Cryptography course, my boss when I was working at the university, and finally my thesis advisor. She provided much help and support to me by holding our bi-weekly meeting thesis discussion sessions, advising me and providing a lot of feedback. I appreciate that very much.

My big gratitude goes to Professor Pekka Orponen, my thesis supervisor. He helped me a lot by letting me work as his research assistant during this time and provided highly valued feedback on my work.

I thank also Dr. Kimmo Järvinen for taking his time and providing helpful feedback.

Special thanks to Jarkko Juntunen and Ari-Pekka Hietala of Lukoton. It was them who initially offered me the idea of this work and made this happen. I appreciate your help during the work and your patience with me being slow sometimes during my time at Lukoton.

Here I want to mention and thank my dear friend Tobias Orth. If not for him I would never find this topic and get to work on this project.

I also would like to thank Dr. Céline Blondeau. She was not working with me on this thesis, but she helped me a lot during my studies and especially being my boss during my summer work at Aalto University.

Now that I mentioned everyone who was directly helping me with this thesis, I would like to thank all people who were around me and helping me just by being there, supporting me in bad moments and cheering me up when I was feeling sad.

First and foremost, my family. Without them I would for sure not have been able to survive these 2.5 years in the Nordics. They have always been ready to help me. I see you only once in a few months now, but that always makes me happier and stronger afterwards.

Of course, my dear friends, here in Finland, back in Russia, and in all different parts of the planet. I have met very many incredible people during the last 2.5 years and they always backed me up when I needed it. I will try to mention everyone now, although it might take a lot of space here.

My sincere gratitude goes to my friends back at home: Alina, Anya, Vera, Dasha, Egor, Zhenya, Katya, Katya, Marina, Misha, Natasha, and Pasha.

My huge thank you also for the people I met already at Aalto. You have made this time truly incredible and tremendous: Alex, Andrei, Andrés, Binghui, Daniel, Daniel, Fabio, Filippo, Gerhard, Elodie, Jan, João, Johannes, Julien, Jules, Marek, Markéta, Misha, Nicholas, Pablo, Ronja, Steffen, and Tobi;

Abhi, Annika, Benno, Carole, Cristian, Daniel, Fernando, Francesco, Georg, Mr Giovanni, Ice, Lando, Leevi, Mareike, Marek, Max, Michelle, Monica, Nikol, Pascal, Philipp, Philippe, Raphael, Rudi, Salvador, Sanya, Sara, Sara, and Stefan;

Adam, Alfredo, Chantal, Charlotte, Clementina, Diogo, Emma, Fabrizio, Frank, Himanshu, James, Jara, Javier, Jessica, Julia, Julian, Kevin, Korkki, Laia, Lilia, Liz, Martina, Mathias, Matias, Minna, Rishabh, Rodion, Santtu, Shino, Tobi, Tom, Unda, and Ziyi;

Andi, Claire, David, David, Fritz, Lauri, Liam, Lukas, Martin, Pieter, Roberto, and Sonja.

My dear Finnish friends, big kiitos and tack to you all and hope to evolve our friendships here: Dan, Juha, Jirko, Jouni, Jussi, Niko, Robert, Stefan, Timo, Tom, and Tuomo.

And last, but not least, my great friends from elsewhere. Thank you and hope to meet you all some day soon: Alberto, Andreas, Ara, Ivo, João, José, Kevin, Rok, Romano, Sasha, and Sasha.

Thank you everyone once again. I am proud and happy that you all have been present in my life, and that it has taken me this much space to mention you all.

Espoo, March 17, 2017

Sergei Chistiakov

Abbreviations and Acronyms

AES	Advanced Encryption Standard
B2B	Business-to-Business
BLE	Bluetooth Low Energy
CBC	Cipher Block Chaining
CFB	Cipher Feedback
CCM	Counter with CBC-MAC
CTR	Counter
ECB	Electronic Codebook
ECC	Elliptic Curve Cryptography
EEPROM	Electrically Erasable Programmable Read-Only Memory
eKey	Electronic Key
FIPS	Federal Information Processing Standard
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IAS	Information Assurance and Security
ID	Identifier
I^2C	Inter-Integrated Circuit
IoT	Internet of Things
IV	Initialisation Vector
JTAG	Joint Test Action Group
MAC	Message Authentication Code
MitM	Man-in-the-Middle
NFC	Near-Field Communication
NIST	National Institute of Standards and Technology
Nonce	Number used once
NTP	Network Time Protocol
OFB	Output Feedback
OOB	Out-Of-Band
PCB	Printed Circuit Board

PKI	Public-Key Infrastructure
RNG	Random Number Generator
SIG	Special Interest Group
SMS	Short Message Service
SoC	System on a Chip
SPI	Serial Peripheral Interface
SRAM	Static Random-Access Memory
SSL	Secure Sockets Layer
SSP	Secure Simple Pairing
URL	Uniform Resource Locator

Contents

Chapter 1

Introduction

The Internet of Things (IoT) is one of the fastest growing fields in Information Technology. People want to make their lives easier, more comfortable and safe by introducing new technologies and devices to their everyday life. One of the examples of IoT integration to our lives is the idea of a smart home. It is a house that can turn the lights on and off automatically, put on your favourite music, make a coffee in the mornings and open the doors for you. And of course the smart home should have a smart lock. You do not need to carry keys anymore, it will open for you automatically.

This thesis is a result of a project of a Finnish company Lukoton Experience Oy. The company develops smart lock systems for Business-to-Business (B2B) industry and already has one system on the market. The author's part in the project was to study the existing system from the security point of view and propose a new improved design, using the embedded cryptographic features of an Electrically Erasable Programmable Read-Only Memory (EEPROM) chip produced by the Atmel Corporation.

The result of this work is presented in this Master's thesis. It is a new design of the Smart Lock system, which keeps the functionality of the previous system the same while incorporating the possibilities offered by the chip to the new system. The thesis will discuss the new design in detail, compare it to the existing system of Lukoton and its analogues on the market.

The following chapters of this chapter will introduce the reader to the thesis in more detail.

1.1 Motivation

As said above, the IoT is a tremendously growing area. "Smart" things become a natural part of our life very quickly. Experts estimate the number

of objects in IoT will reach 50 billion by 2020 [13]. However, people still tend to have conflicting attitudes towards some of the IoT objects. The Smart Lock is a good example of that.

The technique of door locks is based on the same basic idea used for centuries. We still use physical keys to open our doors, as our great-grandparents did in the 19th century. While the technologies for implementing keys and locks have become more advanced and complex, the physical keys still remain, including all their disadvantages. The keys are easy to lose, often hard to find in your pocket or bag, rather complicated and expensive to change or duplicate. After all, they are just one more thing you have to always carry with you. People are used to electronic keys and door locks in offices and hotels, but many still refuse the idea of using the electronic locks at home.

Nevertheless, smart locks slowly but surely incorporate in our usual life. It is still a rather small industry with not many companies working in it. Most of such companies and start-ups concentrate on products for home, while some investigate the B2B market. Lukoton Experience Oy is one of them. It is the first Finnish company working on smart locks for B2B.

The idea of a smart lock system, with electronic keys that can be easily distributed between employees or customers, fits particularly well for businesses. Imagine a company renting out space to third parties in its own storage premises. The usage of physical keys causes many problems and inconveniences. A customer has to come and pick the keys up in the beginning of the lease and return them later on. In case the keys are lost, the company will have to change the lock itself. Finally, the company cannot be sure that the customer did not duplicate the key for any malicious purposes. A smart lock system could immensely simplify the management of a locking scheme. With no physical keys required, the client only needs to install an application on his smartphone, and the company gives him access rights to enter the leased room by adding a record in the database. The keys are easy to duplicate for as many people or smartphones as needed, as well as revoke them afterwards. The system is centralised and can be easily monitored.

Clouds are getting more and more widely used in modern information technologies. A cloud plays also one of the main roles in the Lukoton system. A server placed in a cloud is the centre of the smart lock system, which connects the other entities, smart locks and users to the system and oversees all the processes.

Obviously, security is a core issue in a smart lock system. It requires special attention and thorough analysis, to exclude possible attacks and malfunctions. The system should be lightweight, fast, efficient, and secure at the same time. There is very little information on such systems available at the moment, nearly no specialised literature exists, and the companies working

in this market do not publish their specifications.

1.2 Objectives and scope

This Master's thesis pursues a few objectives. The first one is to capture and analyse the security architecture of the existing smart lock system of Lukoton, investigate its strengths and weaknesses, and decide what to keep the same and what to change.

Secondly, the objective is to examine the proposed EEPROM chip and explore the possibilities it offers to improve the security of the smart lock system. This chip requires scrupulous setting in order to achieve a desired configuration.

The third objective is to propose a new enhanced security architecture for the system, by taking into account major security problems and still keeping it lightweight and suitable for B2B applications.

The fourth objective is to evaluate the security of the proposed design. Some of the most common generic attacks will be reviewed and the system will be tested against them. How does the system resist the possible attacks? The existing system will also be compared with some analogous products on the market. What are the advantages of the new system and where is it the same, or maybe even worse than competitors?

While not a technical objective itself, the overall goal of this Master's thesis is to provide an initial piece of literature in the area of smart lock security to provoke and facilitate future research and work in the field. Being far from complete, this is not a universal handbook for secure smart lock systems design, but presents one solution and some reasoning behind it, which may make it easier for the upcoming work in the area. One can formulate a sub-goal here as explaining details in comprehensible manner, so that even a non-specialist and an average user can get an idea of the basic security principles of a smart lock system.

The scope of the work is the security architecture of a smart lock system, its design, and the use of a chip with strong encryption and authentication capabilities in it. The work does not describe, for example, the functionality of the software either on the smartphone or in the cloud, or the principles of the communication technology and protocols used between the parties of the system.

1.3 Author's contribution

The author of the thesis analysed the existing system, researched the EEPROM chip and proposed a new smart lock system with an enhanced security architecture that enables the system to meet the initial security requirements and challenges. The new system was evaluated and compared to the existing one as well as to some similar products available on the market.

1.4 Structure of the thesis

This Master's thesis is structured as follows. Chapter 2 describes the formal security requirements that the smart lock system has to satisfy. Chapter 3 presents an overview of the existing system design and analyses it from the security point of view. Chapter 4 provides a description of the EEPROM chip, its possibilities, configurations and embedded cryptographic commands. Chapter 5 introduces a new design for the smart lock system and describes the whole procedure step by step from an implementation point of view. Chapter 6 evaluates the new system by comparing it to the existing system, first examining each part of the system, and then discussing its resistance against popular attacks. Chapter 7 overviews the smart lock products available on the market from the security perspective. Chapter 8 compares the new system to the described analogues and indicates its advantages. Chapter 9 concludes the thesis.

Chapter 2

Security Requirements

This chapter defines the requirements for the outcome: which criteria it should satisfy and what distinguishes a desirable result from an undesirable one. We need to establish some formal criteria to evaluate the new system and to compare it with the existing system, as well as with the analogues on the market. This thesis is concentrated on security, hence the requirements will concern the Information Security issues.

When specialists talk about basic principles of Information Security, they often refer to the known triad: Confidentiality, Integrity and Availability. There has been a permanent dispute about adding some other principles to this triad. For instance, a group of researchers made an analysis of Information Assurance and Security (IAS) literature in [9] and proposed the IAS-octave: Confidentiality, Integrity, Availability, Accountability, Auditability, Authenticity/Trustworthiness, Non-repudiation, and Privacy. Another publication by the National Institute of Standards and Technology (NIST) proposed as many as 33 principles [23]. In this work we will take the above-mentioned triad as a base and add a few more requirements, specific for the Smart Lock systems.

2.1 Confidentiality

Confidentiality is a state of information being protected from unauthorised access. It is a basic and intuitive requirement for any security system. Confidentiality requires measures to ensure that only authorised parties will have access to the data.

Confidentiality is usually achieved by using encryption. We need to make sure that all vital information sent between the communicating parties is properly encrypted at any moment of time. Even if an adversary manages

to get parts of the information, it should not anyhow help her to get closer to breaking the whole system and access the rest of the data.

2.2 Integrity

Integrity is a state of information being accurate and having not been changed or tampered with by an unauthorised party. This is another basic principle from the triad described above. Integrity means that a communicating party is sure that the data received is equal to data sent by another authorised party.

Applying that principle to our system, we need to design the system in a way that it is impossible for an attacker to change or corrupt the information without being noticed.

2.3 Authenticity

Authenticity is the ability to verify that the information is produced by an authorised party and to establish which party produced it. Often authenticity is observed as a part of the Integrity principle in the triad, but we separate it to make the criteria clearer.

In the case of the Smart Lock system, we need to make sure that at any moment of time an authorised party knows from whom the received information comes. An adversary cannot impersonate one of the legitimate communicating parties and break the system.

2.4 Security of the key management

Technical accomplishment is done by using cryptographic primitives. They rely on cryptographic keys, which are essential part of the Smart Lock system. Hence, in addition to the three criteria above, we require security of the key management. We need to ensure that the keys are stored and transferred securely between the authorised parties. Only legitimate parties of the system have access to the secret keys, and it is impossible for an attacker to obtain a key stored by the authorised parties.

Chapter 3

Existing System

3.1 Overall picture of the system

This chapter describes outlines of the existing Smart Lock Access Control System. First, we will describe the system in general, and then concentrate on the security part of it. This work will draw an unspecific overall picture of the system, since the main goal of the thesis is to concentrate on security issues. Ulusoy [28] describes the system in more detail.

The Smart Lock Access Control System consists of the following elements: a lock controller, a smartphone, and the cloud. The lock controller includes a microcontroller connected to the electric lock. The microcontroller has a Bluetooth Low Energy (BLE) radio. The smartphone works as the mediator between the lock and the cloud: it receives an encrypted message from the lock and forwards it to the cloud server, and vice versa. The cloud server authorises the smartphone and gives the lock a command to open if the user has access rights.

To register a new user in the system, the cloud server sends an SMS to the phone with a URL link in it. By clicking the link, the user installs user credentials. The credentials include a unique user ID, mobile phone number, and secret information. These parameters are also stored in the server database.

The lock controller and a smartphone communicate with each other via Bluetooth. Communication between a smartphone and the cloud server is organised via HyperText Transfer Protocol Secure (HTTPS). HTTPS is an extension of the Hypertext Transfer Protocol (HTTP) with a connection encrypted by Secure Sockets Layer (SSL) protocol or more advanced Transport Layer Security (TLS) protocol. The smartphone uses the user credentials with secret information to establish a secure connection.

The overall procedure consists of the following steps (see Figure 3.1):

System initialisation

1. The cloud server and the lock share an Advanced Encryption Standard (AES) key.

2. The cloud server sends user credentials to the smartphone over a URL link provided to the phone in an SMS. These credentials include user parameters, such as a unique user ID, mobile phone number, and a secret part. These user parameters are stored in the cloud server database.

3. The smartphone installs the user credentials.

Unlocking process

4. The lock controller creates an AES encrypted advertisement packet and broadcasts it over BLE. When the smartphone gets into the Bluetooth coverage area, it discovers the lock and receives the packet.

5. The smartphone uses secure HTTPS channel to forward the advertisement packet to the cloud server, along with the user credentials.

6. The cloud server extracts the HTTPS protected packet and checks the access rights of the user. Then it decrypts the advertisement packet. If everything is correct, the server authorises the user to open the lock by sending to his smartphone an encrypted message, which contains the virtual key, i.e., permission to open the lock and its validity period.

7. The smartphone gets a timestamp from a Network Time Protocol (NTP) server. If none of the NTP servers in the smartphone's list replies, the timestamp is received from the cellular network.

8. The smartphone forwards the encrypted message along with the timestamp to the lock. If the timestamp is within the unlocking validity period set by the cloud in the encrypted message, the lock switches to the unlocked state. After a certain time it gets locked again.

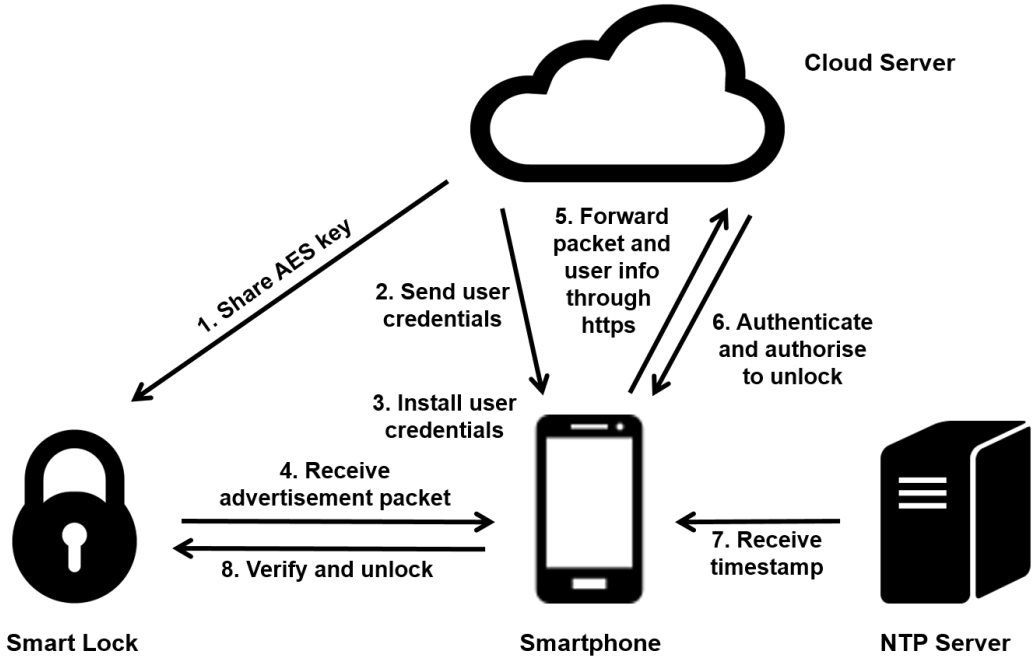

Figure 3.1: Smart Lock Access Control System

3.2 Current security

Now we concentrate on the security used in the working system. As mentioned in the scheme above, the messages sent from the lock to the cloud via the smartphone and back are encrypted. The messages are sent over the Bluetooth Low Energy channel between the lock and the smartphone, and the HTTPS protocol connection is used to send them from the smartphone to the cloud server and back. We will describe these parts of the system security separately.

3.2.1 Message encryption

As seen from Figure 3.1, in Step 4 the lock broadcasts an advertisement packet. This packet consists of: packet length value (1 byte), manufacturer identifier (2 bytes), serial number (2 bytes), battery lifetime level (1 byte), and encrypted data (16 bytes). The last part of the packet is encrypted by the Advanced Encryption Standard (AES) [15] briefly described in Sec-

tion 4.4. The 128-bit long keys are shared between the lock controller and the cloud server during system initialisation. On the lock side the AES key is stored in the Microchip 25A512 Electrically Erasable Programmable Read-Only Memory (EEPROM) [21]. This is a 512-KBit serial EEPROM.

The mode used for encrypting the messages is Cipher Block Chaining (CBC) [24]. Figures 3.2 and 3.3 show the encryption and decryption procedures in this mode respectively.

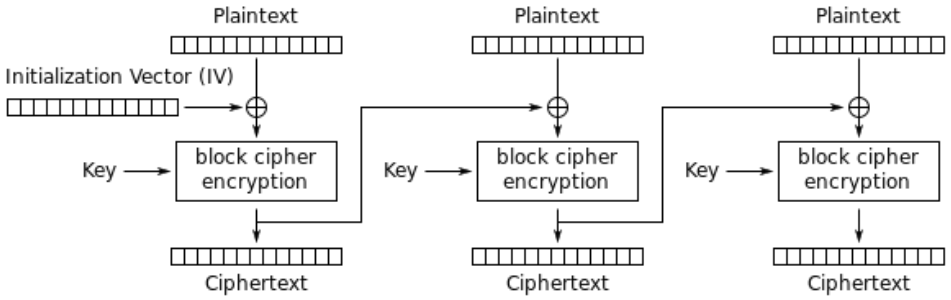

Figure 3.2: CBC mode encryption

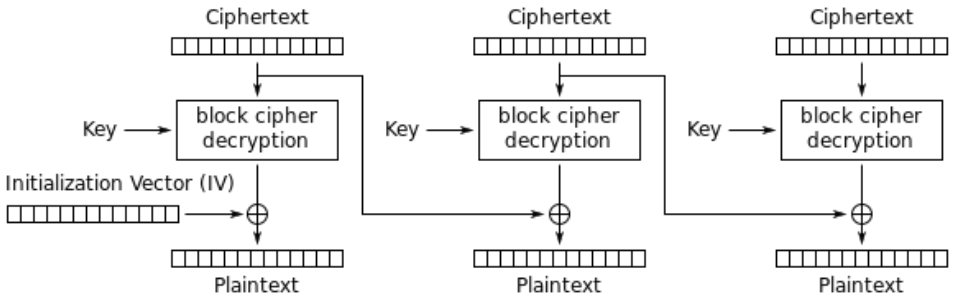

Figure 3.3: CBC mode decryption

The initialisation vector (IV) is also stored in the EEPROM chip for the decryption.

The system also uses an advertisement nonce included in the encrypted data. This nonce is a random number generated by the lock controller for every new advertisement packet. The cloud receives the advertisement nonce in the packet from the lock and includes it in the opening packet. When the lock receives this nonce in the opening packet, it compares it with the one

sent in the previous advertisement packet and only unlocks if both values match.

3.2.2 Bluetooth channel

The Bluetooth channel between the smartphone and the lock may also offer security features. In Bluetooth standard 4.2 [8], the Bluetooth Low Energy protocol uses a Secure Simple Pairing (SSP) [7] model, also referred to as LE Legacy. When using SSP, devices can choose between Numeric Comparison, Just Works, Passkey Entry, and Out-of-Band (OOB) method for pairing. Since the lock controller has no input or output devices, the existing Smart Lock Control Access System can only use the Just Works mode.

Due to the choice of Just Works mode, unauthenticated Diffie-Hellman key agreement is used. It provides protection against passive eavesdropping but not from Man-in-the-Middle (MitM) attacks.

However, the Laird BLE modules that are used in the system do not support any security features of Bluetooth 4.2. Therefore, the system does not use any BLE security features.

3.2.3 Internet channel

A smartphone and the cloud server communicate with each other over the HTTPS protocol [22]. HTTPS provides authentication and protection of the privacy and integrity of the exchanged data. The main goal of HTTPS security is to protect the user credentials and to prevent impersonation of a legitimate smartphone.

3.3 Security analysis

According to Section 3.2.2, we can make a conclusion that between the lock and the smartphone all the security of the system relies on the encryption. The security does not depend on the security of the Bluetooth channel. The whole system was initially designed with the assumption that the messages from the lock to the smartphone and back are sent over an unprotected proximity channel. Such a channel makes the system prone to certain generic attacks, which are described for completeness in Section 6.2.4.

The AES key is only shared between the cloud server and the chip. Even if the attacker captures the packets sent from the lock to the cloud and back, she will not be able to get access to the plaintext data unless she acquires the secret key.

However, it is clear that using only encryption does not make the system secure. An attacker cannot decrypt the messages, but she can intercept an opening message sent from the cloud to the lock, save it and re-use later. The lock will not know that the message is not coming from the cloud. This type of attack is called a Replay attack and will be described in more detail in Section 6.2.3.

The system resists such attacks by using the idea of the advertisement nonce described in Section 3.2.1. The purpose of the advertisement nonce is to avoid Replay attacks and to achieve authentication. Only the cloud can decrypt an advertisement packet, so the lock gets some evidence that if an opening message contains the same advertisement nonce, then it comes from the cloud server. The advertisement nonce is generated fresh for each new advertisement packet.

Additionally, the opening message sent from the cloud back to the lock (via the smartphone) is valid only for a limited time. The message becomes invalid after one unlocking event. That also makes Replay attacks more difficult.

Possible impersonation of the legitimate smartphone is prevented by using HTTPS security between the smartphone and the cloud server. When registering in the system, the smartphone has to install the user credentials sent by the cloud server. These credentials are used for authenticating the smartphone to the cloud server.

An obvious flaw in the system is insecure storage of the keys in the EEPROM chip. The chip is not physically protected against acquiring the data stored in it. Hence, if an attacker manages to get access to the chip and get the AES secret key, the system security for this lock will be broken.

Another security issue is the lack of message integrity. The parties cannot detect if the messages have been tampered with due to the lack of a Message Authentication Code (MAC).

Chapter 4

The Atmel AES132A Chip

4.1 Introduction

Electrically Erasable Programmable Read-Only Memory (EEPROM) is a memory type that allows to erase and reprogram the bytes stored there. EEPROM chips have been used quite widely in many different devices that do not require storing big amounts of data.

However, the modern world requires technologies to change and develop very fast. The security part of the products, often neglected in the recent past, has now become an essential detail. People are well aware how easy it can be nowadays to break a whole system using the security flaws. In order to be competitive on the market, technology products of all kinds have to provide good security. The EEPROM chip market was not an exception, and companies started to design and produce secure chips with embedded cryptography. One of these companies is Atmel. In this chapter we will describe the Atmel AES132A chip, which is an essential part of the Smart Lock Access Control System.

The ATAES132A chip provides both authentication and confidential data storage. The chip has 32Kb of Standard Serial EEPROM memory, 16 slots for 128-bit keys, and 24 embedded cryptographic functions. The chip uses the Advanced Encryption Standard in Counter with CBC-MAC mode (AES-CCM), where CBC-MAC stands for a Cipher Block Chaining Message Authentication Code. The Atmel AES132A chip provides also a high level of physical security, effectively preventing accessing the confidential data stored on the chip.

4.2 Components of the EEPROM

The ATAES132A EEPROM is divided into several segments. Each of them has different functions. These memory segments are the following:

- User Memory: contains 32Kb for data storage purposes.
- Key Memory: contains 256 bytes for storing the secret keys used in different cryptographic functions.
- Configuration Memory: contains the configuration information, security control registers, and counters.
- SRAM Memory: used to store the volatile data and status information.

We will describe the abovementioned memory segments in more detail in the following sections.

4.2.1 User Memory

The User Memory has 32Kb space, which is split into 16 user zones of 2Kb each. Each user zone is associated to a specific User Zone Configuration Register in the Configuration Memory and can be set separately. The possibilities of configuring a user zone include authentication requirement, Read Encryption, and Write Encryption. These security requirements can be combined freely. It is also possible to disable all the security requirements. A user zone can be accessed only if all set security requirements are satisfied. All the bytes within one user zone have the same access restrictions.

4.2.2 Key Memory

The Key Memory has 16 slots for the secret keys, each 128 bits long. Each key is associated to a specific Key Configuration Register in the Configuration Memory. The Key Configuration Register defines which cryptographic functions can use the key. Also, the register can be configured to require prior authentication to access the key.

The Key Memory cannot be read under any circumstances. Writing in the Key memory is possible using the `Write`, `EncWrite`, `KeyCreate`, `KeyImport`, `KeyLoad`, and `KeyTransfer` commands. The `Write` command performs writing cleartext data into the key memory and is only possible before locking the chip. The `EncWrite` performs encrypted writes; it is possible to use after locking the chip if this is allowed in the Key Configuration register. The last four commands manage the key registers after the locking.

4.2.3 Configuration Memory

The Configuration Memory contains all the registers defining restrictions for access and usage of different parts of the memory. These include user zone access requirements, the key usage restrictions, and the counter usage restrictions. Device-level configuration option registers as well as several registers for customer information are also located in the Configuration Memory.

The Configuration Memory can always be read using the `BlockRead` command. The content of the Configuration memory is permanently locked by using the `Lock` command.

The list of possible configuration settings for the User Zone, Key, VolatileKey, and Counter registers can be found in Appendix A.

4.2.4 SRAM Memory

The Static Random-Access Memory (SRAM) is used to store volatile data and status information. The SRAM buffers and registers are located on top of the memory address space. The SRAM is used to store the following elements:

- Nonce: used by different cryptographic functions.
- Pseudo Random Number Generator (RNG) seed.
- VolatileKey: a session key register. Can only be used for authentication and to encrypt or decrypt external data.
- Command memory buffer: used to send extended commands to the device.
- Response Memory Buffer: used to read responses to the extended commands from the device.
- IO Address Reset Register: used to reset the buffer address pointers.
- STATUS Register: reports the state of the device.
- Authentication Status Register: stores the result of most recent authentication attempt.

4.3 Physical security

It is also very important to provide security not only at software level, but also at hardware level. Nowadays it is not complicated to find a microprober or conduct a power analysis attack using an oscilloscope. The ATAES132A chip has various physical security mechanisms that prevent an attacker from getting the values of secret keys stored in the memory [4]. Following are some of the features providing hardware security in the chip:

- Active shield covering the entire chip.
- Internal memory encryption.
- Constant time/cryptographic operation pattern.
- Randomised mathematical operations.
- Internal state consistency checking.
- Voltage tampering detector, isolated power rail.
- Temperature tampering detector.
- Internal clock generation.
- Secure test methods, not the Joint Test Action Group (JTAG) [18] ones.
- Frequency tampering detector.
- No debug probe points, no test pads.
- No package or die identification.

The chip is designed to prevent or significantly complicate most side-channel attacks. Some examples of such attacks are the following:

- Microprobe attacks.
- Timing attacks.
- Emissions attacks.
- Faults, invalid command attacks.
- Clock glitches.

4.4 Advanced Encryption Standard

The chip uses the Advanced Encryption Standard (AES), the current cryptographic standard encryption function recommended by the U.S. National Institute of Standards and Technology (NIST). The cipher was developed by Joan Daemen and Vincent Rijmen, under the name of Rijndael [10] and submitted as a proposal to NIST during the AES selection process. In 2001 NIST approved the cipher to be a standard of encryption [15].

AES is a substitution-permutation network, working with 128-bit long blocks and three possible key lengths: 128, 192 or 256 bits. The encryption process comprises 10, 12 or 14 rounds, depending on key size. Each round consists of four steps: `SubBytes`, `ShiftRows`, `MixColumns` (not performed in the last round), and `AddRoundKey`. The ATAES132A chip uses 128-bit long keys.

Common modes of operation for AES include Electronic Codebook (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), Output Feedback (OFB), Counter (CTR). CBC and CTR are the modes used most often; they are recommended, for example, by Niels Ferguson and Bruce Schneier in [14].

The ATAES132A chip uses the AES in CCM (Counter with CBC-MAC) mode [25]. As follows from the name, the mode combines the Counter mode for encryption with CBC-MAC (Cipher Block Chaining Message Authentication Code). Cleartext MAC is computed first on the given message, using the chip specific data and the plaintext as inputs in CBC mode. Then the plaintext and the cleartext MAC are encrypted in the Counter mode. The same key is used for both operations.

The CCM mode provides authenticity, integrity, and confidentiality. Due to the CCM being a derivative of the Counter mode, which is, in fact, a stream cipher, the nonce should not be used more than once for one key.

Figures 4.1 and 4.2 show the flow of the CCM mode on an example of the `Encrypt` and `Decrypt` commands.

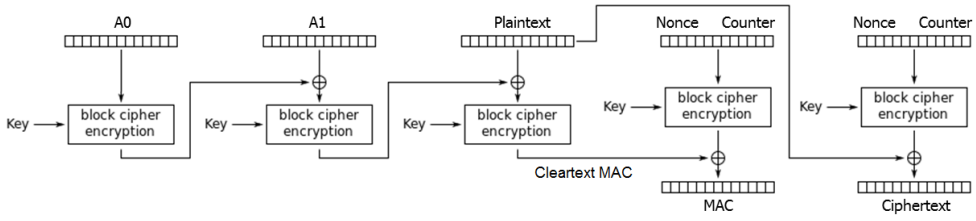

Figure 4.1: CCM mode encryption

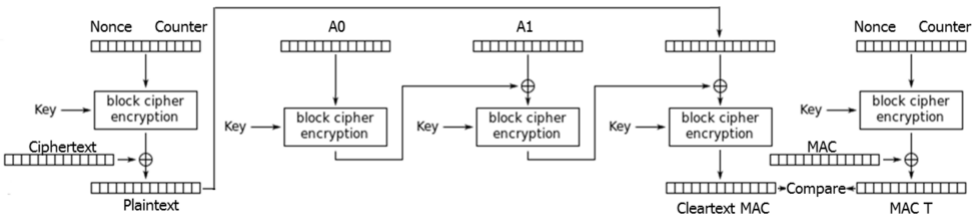

Figure 4.2: CCM mode decryption and MAC verification

The CCM mode allows to encrypt and decrypt any number of 128-bit blocks of plaintext, but the figures above illustrate the case with input of one block, as it is the situation in the actual system. A0 and A1 are chip specific data including, for example, the length of a plaintext, parameters of the `Encrypt` or `Decrypt` command, and a nonce. They are needed for creating a MAC. The same nonce is used in the right side of the figures. The input there consists of a random number, a MacCount value (considered as a part of the CCM nonce in this case), and a counter, as usual in CTR mode. The random number and MacCount are discussed in more detail in Section 5.2.

4.5 Commands

Apart from the standard EEPROM `Read` and `Write` commands, the chip has 24 different cryptographic functions. The full list of commands with their short descriptions is presented in Appendix B.

In this part of the work we describe in detail the commands used in the Smart Lock Access Control System, namely `Read`, `Write`, `BlockRead`, `Decrypt`, `Encrypt`, `KeyCreate`, `Lock`, `Nonce`, and `Random`.

4.5.1 `Read` command

The `Read` command reads the requested number of bytes starting from the specified point in the memory and returns the data to the Host.

When the `Read` command is received, the device checks the AuthRead and EncRead bits in the ZoneConfig Register for the user zone. In case the bytes requested are not allowed to be read, `0xFF` will be returned. The command may be used to read any number of bytes in one operation. It can also cross the EEPROM page boundaries.

Reading the Key Memory is never allowed. The `Read` command cannot read the Configuration Memory, the `BlockRead` command should be used instead.

4.5.2 `Write` command

The `Write` command writes 1 to 32 bytes to the EEPROM starting from the specified point in the memory.

The command cannot write data to different user zones in one operation. The data to be written cannot cross the EEPROM page boundaries.

Writing to the Key Memory or the Configuration Memory is only allowed before locking. Partial writes to key registers are not allowed.

4.5.3 `BlockRead` command

The `BlockRead` command reads 1 to 32 bytes of plaintext data from a User Zone or the Configuration Memory. It also returns an error code in case the Read was not successful.

The `BlockRead` command can only read data from a single User Zone at one attempt; it cannot cross the EEPROM page boundaries. The command cannot read the Key Memory under any circumstances.

The possibility to read a User Zone depends on the values of the EncRead and AuthRead bits in the configuration settings of a User Zone (see Table A.1).

4.5.4 `Decrypt` command

The `Decrypt` command takes 16 to 32 bytes of ciphertext data, decrypts it with the specified key, stored in the Key Memory, verifies the MAC, and returns the encrypted data to the Host if the MAC matches.

The command requires a valid nonce to be run. Depending on the configuration, the nonce must be or need not be random. Prior authentication may be required if needed. Also the values of the Usage Counter, SerialNum, and SmallZone may be included to the MAC computation.

Two modes of the command are possible:

- Client Decryption mode: decrypts data encrypted by the same or another ATAES132A device.
- Normal Decryption mode: decrypts data encrypted by a cryptographic Host. In this mode it cannot decrypt data encrypted by an ATAES132A device.

In the Client Decryption mode the following requirements must be satisfied:

- The encrypting and decrypting devices must both contain identical keys.
- The decrypting device must know the KeyID used for encryption. It is passed to the decrypting device as one of the parameters of the command.
- The decrypting device must know the nonce used for encryption. The nonce is passed to the decrypting device using the `Nonce` command or synchronised on both devices using the `NonceCompute` command.
- The decrypting device must know the number of bytes encrypted. It is passed to the decrypting device as one of the parameters of the command.
- The decrypting device must know the MacCount used for encryption. It is passed to the decrypting device as one of the parameters of the command.
- Mode bits responsible for including the values of the Usage Counter, SerialNum, and SmallZone to the MAC computation must be the same on both encrypting and decrypting devices.

4.5.5 `Encrypt` command

The `Encrypt` command takes 1 to 32 bytes of plaintext data, encrypts it with the specified key, stored in the Key Memory, and returns the encrypted data to the Host along with a generated MAC.

The command requires a valid nonce to be run. Depending on the configuration, the nonce must be or need not be random. Prior authentication may be required if needed. Also the values of the Usage Counter, SerialNum, and SmallZone may be included to the MAC computation.

4.5.6 `KeyCreate` command

The `KeyCreate` command generates a 16-byte long random number, stores it either in the Key Memory or in the VolatileKey Register, and returns the key value encrypted with the Parent Key to the Host, along with a generated MAC.

The key configuration contains a link to the slot number where the parent key is located. The command requires a valid nonce to be run. Depending on the configuration, the nonce must be or need not be random. Prior authentication may be required if needed. If the target of the command is the VolatileKey Register, then an InMAC must be provided.

The command allows rather flexible mode configuration. The user decides whether to store the newly generated key in the Key Memory or in the VolatileKey Register, and if the RNG Seed needs to be updated before the key generation. Also the values of the Usage Counter, SerialNum, and SmallZone may be included to the MAC computation.

4.5.7 `Lock` command

The `Lock` command permanently locks different segments of the chip memory. These segments are the Configuration Memory (also locking Key, Counter and Use Memory access restrictions), the Key Memory, and the SmallZone register.

The Configuration memory must be locked prior to locking the Key Memory.

The command writes `0x00` to the specified lock register in order to lock the segment of the EEPROM. In an unlocked state these registers contain `0x55`. The Configuration Memory is controlled by the LockConfig Register, the Key Memory – by the LockKeys Register, and the SmallZone Register – by the LockSmall Register.

One of the optional parameters of the `Lock` command is a checksum (CRC-16) calculated over the memory segment being locked. If the value of the checksum entered as the parameter matches the value calculated by chip, the operation is succeeded; otherwise the command returns an error code.

4.5.8 `Nonce` command

The `Nonce` command generates a 96-bit long nonce, stores it in the SRAM Nonce Register, and optionally returns the random number used for the nonce generation.

Many other cryptographic commands require a valid nonce. The user has to provide a 96-bit long InSeed value and decides whether the RNG Seed needs to be updated before the key generation.

There are two possible modes for running the command. These modes are the following:

- Inbound Nonce: the InSeed value set by the Host is used as a nonce;

no cryptographic calculations are performed. The command does not return any output to the Host in this mode.

- Random Nonce: an internal pseudo RNG generates a random number, which is cryptographically combined with the InSeed value provided by the Host. The 16-byte random number is returned to the Host.

There is no need to run the command before every cryptographic operation, because the chip automatically includes MacCount in the MAC calculations. This guarantees uniqueness of a nonce. However, the maximum value of MacCount is 255, so one has to run the `Nonce` command at least after every 255 cryptographic operations performed by the chip. There are several other events that make a nonce invalid. The nonce is valid until one of the following happens:

- MacCount reaches the maximum value of 255.
- A MAC comparing operation fails.
- The cryptographic state machine is reset due to either receipt of a `Reset` command, power cycling, or activation of the initialisation sequence due to Wake-up from the Sleep power state.
- The `Nonce` command is run again, setting MacCount to zero.

4.5.9 `Random` command

The `Random` command generates a random number and returns it to the Host. The chip uses the internal high-quality pseudo RNG and the random number generation procedure recommended by NIST [26]. There are two possible modes for running the command. These modes are the following:

- Random Number Generation: the generated 16-byte random number is not stored internally and only returned to the Host.
- Nonce Synchronisation: in addition to returning the generated 16-byte random number to the Host, the first 12 bytes are stored in the Nonce Register for later use by the `NonceCompute` command.

Chapter 5

New System

5.1 Introduction of security to the system

The existing system described in Chapter 3 has some security issues as described in Section 3.3. One of the problems is lack of message integrity. The cloud server and the chip cannot be sure that the message received has not been tampered with on the way. Another issue is that the current EEPROM chip is used to store the data on the lock side. It does not provide any secure storage, so it is relatively easy to get access to the data stored there, including the AES secret key used for encryption of the data in the advertisement packet. The main goal of this work was to design a way to fix these security flaws and implement the corrections in the existing system.

As discussed in Chapter 4, an essential part of providing security to the system is using the AES132A chip. It allows to flexibly and efficiently set up the necessary security in the system. It uses AES encryption in a standard mode, recommended by NIST, which additionally provides authentication and message integrity by using the MAC. Also the chip is protected on both the hardware and the software side. The chip manufacturer guarantees that the keys stored in the Key Memory cannot be read under any circumstances. We used the opportunities provided by the chip in order to establish a secure Smart Lock Access Control System.

We will rewrite the general flow of the access control procedure by adding some security steps. The new procedure consists of the following steps (see Figure 5.1):

System initialisation

1. The cloud server generates a random 128-bit number that is written into the Key Memory of the chip. This number will be the shared AES

key between the chip and the cloud.

2. The chip is configured and locked.

3. The chip populates the rest of the slots in the Key Memory with random numbers.

4. The cloud server sends user credentials to the smartphone over a URL link provided to the phone in an SMS. These credentials include user parameters, such as a unique user ID, mobile phone number, and a secret part. The user data is stored in the cloud server database.

5. The smartphone installs the user credentials.

Unlocking process

6. The chip creates an advertisement nonce and includes it in an advertisement packet. The chip encrypts this packet and adds MAC using AES in CCM mode. Then the packet is broadcasted over BLE. When the smartphone gets into the Bluetooth coverage area, it discovers the lock and receives the packet.

7. The smartphone uses secure HTTPS channel to forward the advertisement packet to the cloud server, along with the user credentials.

8. The cloud server extracts the HTTPS protected packet and checks the access rights of the user. Then the cloud server verifies the MAC and decrypts the advertisement packet with the shared AES key. If everything is correct, the server authorises the user to open the lock by creating a new opening message with a validity period for unlocking, adding the received advertisement nonce, encrypting it with the same shared AES key, adding the MAC and sending it to the smartphone.

9. The smartphone gets a timestamp from a Network Time Protocol (NTP) server. If none of the NTP servers in the smartphone's list replies, the timestamp is received from the cellular network.

10. The smartphone forwards the encrypted opening message along with the timestamp to the lock. The chip verifies the MAC and decrypts the opening message with the shared key, checks if the received advertisement nonce equals the one originally created by the chip. Then the chip checks whether the timestamp is within the unlocking validity period sent by the cloud. If everything is correct, the lock switches to the unlocked state. After a certain time it gets locked again.

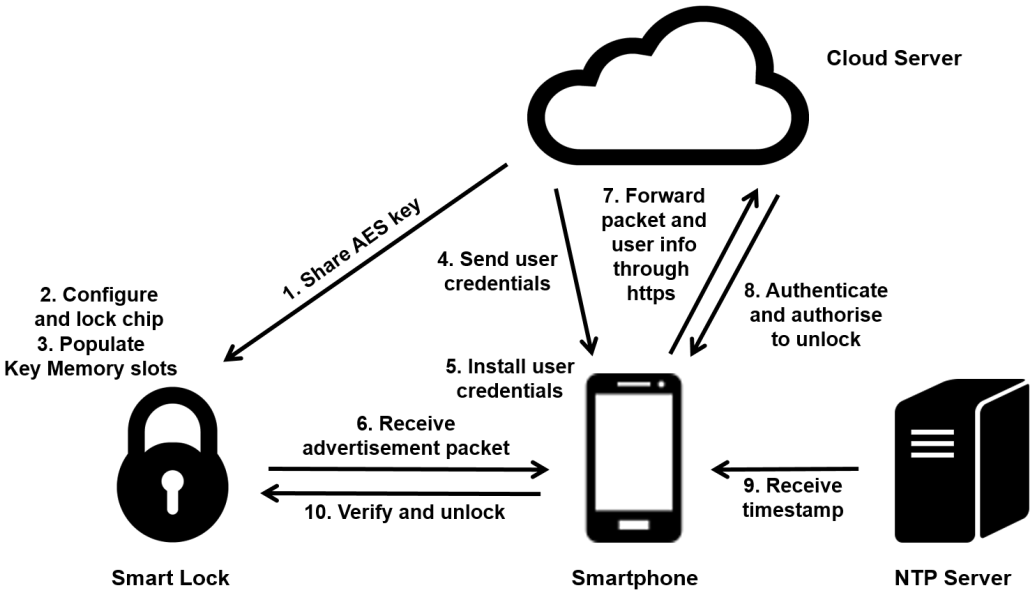

Figure 5.1: Smart Lock Access Control System

5.2 Implementation options

The actual implementation process brought up many detailed issues to think about. One of the most important ones was the question about a nonce used for encrypting the message and calculating the MAC in AES-CCM mode. To verify the MAC and decrypt the received message, one needs to know the 13-byte nonce (called CCM nonce further in the text) used for encryption. This nonce consists of a 12-byte random number and a 1-byte MacCount. The existing system uses a fixed IV for CBC encryption mode, but that is not possible for CCM mode. Using a fixed nonce would allow an attacker to easily get XORs of the plaintexts because the encrypted CCM nonce would stay constant (see Figure 4.1). A solution with predictably changing the CCM nonce (for example, incrementing a nonce by one for each new operation) breaks in case of an opening message delivery failure: the cloud server would try using an incremented CCM nonce for the next operation, while the chip would apply the same one again. Some resynchronisation process must be applied. Therefore, the problem of sending a CCM nonce along with the message to decrypting party arises. Two implementation options were proposed. They are referred to as the Chip-based solution and the Cloud-based

solution in the following sections.

5.2.1 Chip-based solution

This solution is called Chip-based, because the responsibility for generating a CCM nonce for each new message is on the chip. In this method the chip generates a new CCM nonce using its integrated Random Number Generator and attaches it in clear to each advertisement packet. The cloud server uses the received CCM nonce to verify the MAC and decrypt the message. The same CCM nonce will be used by the cloud server to encrypt an opening message and calculate a MAC for it.

The chip uses an idea of MacCount. This is a one byte counter increasing by one from 0x00 to 0xFF every time some command uses the CCM nonce stored in the chip SRAM. This allows to keep the CCM nonce fresh and at the same time avoid generation of a new random number for every new advertisement packet. The chip only needs to do it after each 256 operations. In addition, this gives the possibility to use the same random number for an opening message for the cloud server. The cloud would only need to increase the MacCount value (received in the advertisement packet) by one. The cloud then does not need to send the CCM nonce back to the chip because it is already stored in the chip's memory for decryption.

Figure 5.2 illustrates the flow of this procedure.

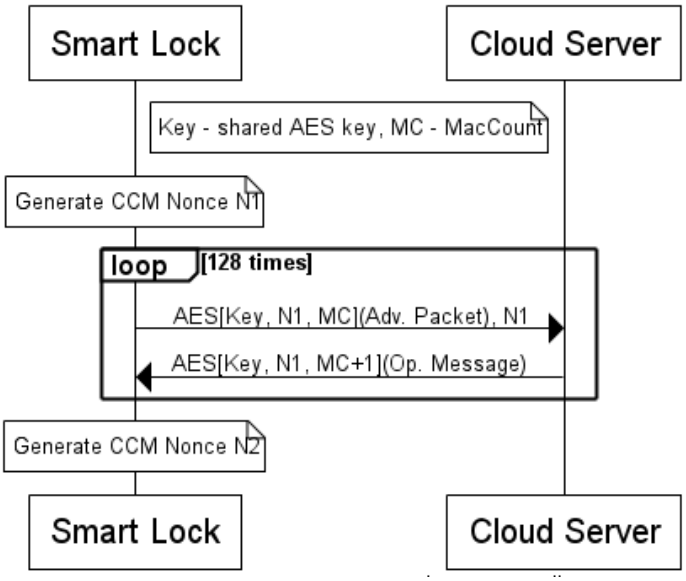

Figure 5.2: Chip-based solution

5.2.2 Cloud-based solution

The second solution proposes moving the responsibility for the random part of the CCM nonce generation from the chip to the cloud side. In this scenario the cloud server generates a new random number for the CCM nonce using its own RNG for every new opening message. This new nonce is attached in clear to the opening message sent to the chip. The chip uses the received CCM nonce for verifying the MAC and decrypting the message. The same nonce is re-used for encrypting the next advertisement packet and calculating a MAC for it.

For the very first advertisement packet the chip cannot use the CCM nonce received from the cloud server due to the lack of previous correspondence with it. The chip uses its unique Serial Number as an initial CCM nonce. The cloud server stores the Serial Number of each chip, so it will be able to verify the MAC and decrypt the first advertisement packet.

MacCount is almost not used in this solution. Every time a new CCM nonce is introduced to the chip, the MacCount value automatically falls down to 0x00. However, the MacCount value still keeps nonces used for an opening message and a following advertisement packet different. Due to the fact that a new random number is generated for every opening message, its value will always be 0x01 for any opening message and 0x02 for the succeeding advertisement packet.

This solution also uses a backup mechanism of storing a previous CCM nonce in the memory of both the chip and the cloud server. In case of an opening message corruption (due to the actions of an adversary or for other reasons), the chip would use the previous CCM nonce for the next advertisement packet. The backup CCM nonces are updated in the memory of both the chip and the cloud server after each successful message decryption.

Figure 5.3 illustrates the flow of this procedure.

5.2.3 Analysis of the proposed solutions

Both described solutions satisfy the security requirements and effectively prevent an adversary from breaking the encryption and integrity. The CCM nonces used for encryption and MAC calculation are updated every time, which prevents possible Replay attack (the attack described in more detail in Section 6.2.3). However, there are differences from the implementation and security points of view.

First difference is about to which message a CCM nonce is attached to. The Chip-based solution has it sent along with an advertisement packet, while the Cloud-based solution sends it along with an opening message.

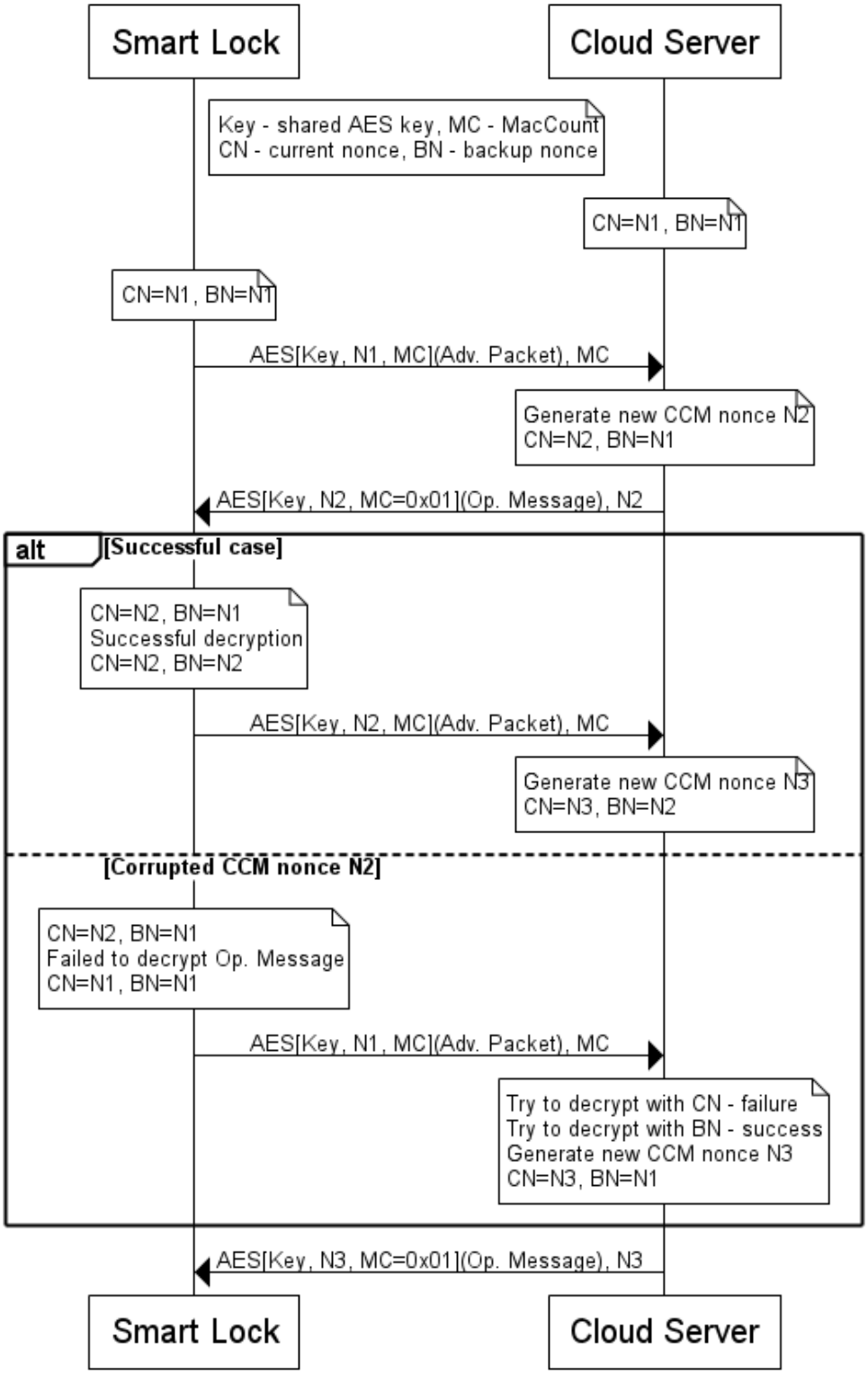

Figure 5.3: Cloud-based solution

But the chip has to receive an opening message only once, while it has to broadcast an advertisement packet constantly. So, the Cloud-based solution allows to keep an advertisement packet length shorter, which is rather a big advantage in terms of energy saved for the chip.

Both solutions assume sending a CCM nonce in clear. This does not lead to a breach of security, because a nonce, essentially, does not need to be secret.

Another advantage of the Cloud-based solution is that it saves the resources of the chip RNG. The chip already has to produce an advertisement nonce for every message and additional CCM nonce generation will result in decrease of the chip lifetime. This happens because the chip has limitations in quantity of random numbers generated. However, this advantage is not too big, since in the Chip-based solution a new random number for the CCM nonce has to be produced only once for 128 advertisement packets (another 128 MacCount values are spent on the opening messages decryption and verifying the MAC). It should be mentioned also that the quality of the RNG in the cloud server is much higher, due to the limitations of small EEPROM chips.

The Cloud-based solution requires to generate a random number 128 times more often, but the responsibility for it falls on the cloud RNG, resources of which are considered unlimited for the purposes of this work. Overall, the Cloud-based solution requires more steps and is more complicated.

A possible attack on the Cloud-based solution by corrupting the CCM nonce sent from the cloud server to the chip is prevented by the backup mechanism described in the previous section. In case of a corrupted received nonce, the chip will use the previous "successful" CCM nonce stored in its memory for the next advertisement packet. The cloud server also stores the same CCM nonce, so it will be able to verify the MAC and decrypt the received packet.

From the cryptographic point of view, the Cloud-based solution offers one significant improvement by providing additional authentication of the chip. When the cloud server receives an advertisement packet in the Chip-based solution, it does not know whether it is a message from the chip or not. An adversary can replay some of the previous advertisement packets and the cloud will generate an opening message for it. This opening message, nevertheless, will not be accepted by the chip because the advertisement nonce inside it will have expired by then. On contrary, in the Cloud-based solution the cloud server will reject a previously used advertisement packet, because the CCM nonce for it will not match the one stored in the cloud server database. This also saves resources of the cloud server, since it will

not need to generate a new opening message in this case.

According to the above, the Cloud-based solution has more advantages over the Chip-based solution. Hence, we adopt the Cloud-based solution to be used and described in the next section.

5.3 Implementation

This part of the work describes the security steps of the Smart Lock Access Control System in detail. In this scenario, a company rents out some rooms to a third-party customer. The access to these rooms is secured by the system discussed. As said above, the Cloud-based solution is used in the description.

5.3.1 Key sharing

In order to start the secure data exchange between the lock and the cloud, the secret key must be shared between them. This is done by the manufacturer in the system initialisation step, before passing the system to the company. The random number is created by the cloud server and manually written into the Key Memory of the chip. This number will function as the secret key shared only between the chip and the cloud server (referred to as the "shared key" in this chapter). The key is not otherwise stored anywhere. These steps are summarised as follows:

1. The cloud server generates a random 128-bit number.
2. This number is written into the Key Memory of the chip using the `Write` command.
3. The number is saved in the cloud server database along with the Serial Number of the chip.

5.3.2 Chip configuration and locking

Before sending it to the company, the chip should be configured appropriately. The ATAES132A chip provides very flexible configuration settings for User Memory, Key Memory, Volatile Key and Counter. The configurations follow the policy of allowing only necessary operations. After configuring all the memory zones, the chip has to be locked. First the Configuration Memory is locked, then the Key Memory, and lastly the Small Zone Register. These steps are summarised as follows:

1. Configure all the User Memory slots.
2. Configure all the Key Memory slots, including the one with the shared secret key.
3. Configure the Volatile Key Register.
4. Configure the Counter Registers.
5. Lock the Configuration Memory using the `Lock` command.
6. Lock the Key Memory using the `Lock` command.
7. Lock the Small Zone Register using the `Lock` command.

5.3.3 Populating the Key Memory

This step is also done by the manufacturer before sending the product to the customer. Now it must be ensured that the keys stored in the other slots of the Key Memory cannot be used by an attacker or a third party. Default key value initially set by Atmel in all the key slots is `0xFFF...FF`. The commands addressing the Key Memory can specify a KeyID different from the one with the shared key. So, an attacker might use it. In order to avoid that, the rest of the slots are populated using the `KeyCreate` command. The command generates random numbers that are written into the Key Memory. The command returns an encrypted value of the just generated key, but it is not in any way processed further or stored anywhere, so no one will know the actual key values in these 15 key slots. Hence, the possibility of using them to access the chip is excluded. These steps are summarised as follows:

1. Create a nonce using the `Nonce` command.
2. Populate the other 15 slots in the Key Memory with random key values using the `KeyCreate` command. Do not process the output of the command.

5.3.4 Creating an advertisement packet

This is the first step after the system initialisation. The chip needs to create an advertisement packet that will be broadcasted over Bluetooth, waiting for a user's smartphone to receive and accept the packet. The advertisement packet consists of: packet length value (1 byte), manufacturer identifier (2 bytes), serial number (2 bytes), battery lifetime level (1 byte), and encrypted

data (16 bytes). The encrypted data contains an advertisement nonce generated by the chip, which is also stored in the User Memory. A data encryption and a MAC calculation are done using the shared key and a CCM nonce stored in the chip memory. These steps are summarised as follows:

1. Create a random number using the `Random` command for using as an advertisement nonce.
2. Write this nonce into the User Memory using the `Write` command.
3. Encrypt 16 bytes of data including the advertisement nonce and system specific data and calculate a MAC using the `Encrypt` command with the shared key and a CCM nonce stored in the chip memory.
4. Form an advertisement packet consisting of packet length value (1 byte), manufacturer identifier (2 bytes), serial number (2 bytes), battery lifetime level (1 byte), and encrypted data (16 bytes).

5.3.5 Operations on the cloud side

The cloud receives an advertisement packet and the user information from the smartphone over the secure HTTPS channel. First, the cloud server extracts the packet and checks the user's access rights. If the user is permitted to unlock the lock, the cloud server verifies the MAC and decrypts the encrypted data in the advertisement packet. The shared key and a current CCM nonce stored in the cloud server database are used for decryption. The cloud server checks all the data in the decrypted message for correctness. Then the cloud server generates a random number for a new CCM nonce and constructs a new opening message, including the validity period when the lock can be unlocked and the advertisement nonce, received from the chip. This opening message is encrypted and protected by MAC using the same shared key and just generated CCM nonce. Then the cloud uses HTTPS channel to send the encrypted opening message along with the new CCM nonce to the smartphone. These steps are summarised as follows:

1. The cloud server receives an advertisement packet with encrypted data and the user information from the smartphone over the HTTPS channel.
2. The cloud server extracts the packet and checks whether the user has access rights to the lock.

3. The cloud server verifies the MAC and decrypts the encrypted data in the advertisement packet using the shared key and the CCM nonce stored in the server database.

4. The cloud server checks the decrypted data for correctness.

5. The cloud server generates a random number to be used in a new CCM nonce and stores it in its database.

6. The cloud server creates a new opening message, including the validity period when the lock can be unlocked and the advertisement nonce, received in the encrypted data of the advertisement packet.

7. The cloud server encrypts the opening message and calculates a MAC with the shared key and just generated CCM nonce.

8. The cloud server sends the encrypted data along with the new CCM nonce to the smartphone over the HTTPS channel.

5.3.6 Unlocking the smart lock

First, the lock receives a timestamp from a smartphone and sends a notification of reception to the phone. After that the smartphone sends the encrypted opening message and the CCM nonce that it got from the cloud server. The chip verifies the MAC and decrypts the opening message using the shared key and received new CCM nonce. The chip checks whether the advertisement nonce in the decrypted message matches the advertisement nonce originally created by the chip and stored in the User Memory. Then the chip checks if the received timestamp fits in the unlocking validity period defined by the cloud server and stored in the message. If all the checks are successful, the lock sends a notification to the smartphone and unlocks for a certain time. These steps are summarised as follows:

1. The smartphone sends a timestamp to the lock controller.

2. The lock controller sends a reception notification to the smartphone.

3. The smartphone sends the encrypted opening message, also including the CCM nonce to the lock controller.

4. The chip verifies the MAC and decrypts the opening message with the shared key and received CCM nonce using the `Decrypt` command.

5. The chip compares the advertisement nonce received in the opening message with the advertisement nonce stored in the User Memory.

6. The chip checks whether the timestamp value is in between *OpeningTime* and *OpeningTime+OpeningDuration*, where *OpeningTime* is the time value when the lock can be unlocked and *OpeningDuration* is the duration of the opening window. Both values are provided by the cloud server in the opening message.

7. The lock controller sends a notification of acceptance to the smartphone.

8. The smart lock is unlocked.

Chapter 6

Evaluation

6.1 System parts comparison

This chapter will compare the existing system described in Chapter 3 with the proposed one described in Chapter 5 from the security point of view. We will point out and discuss the similarities and differences and also later discuss how the systems react on some well-known attacks. In this chapter we will refer to the two compared systems as the "existing system" and the "new system". We start with comparing parts of the systems with each other in the following sections.

6.1.1 EEPROM chips

The existing system uses Microchip 25A512 EEPROM, and the new system exploits the Atmel AES132A device. The former EEPROM chip has much larger storing capacity: 512Kb as compared to 32Kb of the latter one. However, the Smart Lock Access Control System does not require much space to store data on the lock side, since it is the cloud that stores all the log and other information. The EEPROM chip just needs to store the shared key and the nonce values.

The Atmel chip is much better protected physically. As described in Section 4.3, it has many features providing hardware security of the chip. Some examples of these features include an active shield covering the entire chip (while usually only parts of the chips are protected this way), detectors for voltage, temperature, light or frequency tampering, and internal memory encryption. The Microchip EEPROM provides much fewer features for physical chip protection.

Some other important characteristics of the EEPROMs (writing endurance, data retention, supported temperature range) are similar in both chips.

6.1.2 Storage security

As discussed in the previous chapters, storage security is an obvious advantage of the new system. The Atmel AES132A chip provides 16 secure storage slots for the secret keys, which can not be read under any circumstances (see Section 4.2.2). The existing system suffers from the possibility of an attacker extracting the key values from the chip, while the new system excludes this scenario.

The Atmel chip also has much better flexibility in configuring the access rights to different memory slots of the chip. For example, a memory slot in the User Memory can be set in a way that reading/writing data requires prior authentication and/or decryption/encryption with the specified key. The user can apply different access rights configurations for each of the 16 zones in the User Memory. The Microchip EEPROM lacks such flexibility.

6.1.3 Encryption

Both the existing and the new systems use AES encryption with 128-bit keys. The difference is in different modes of operation: the existing system uses CBC mode while the new one uses CCM mode (see Sections 3.2.1 and 4.4). Both are recommended by NIST [24]. The CCM mode is essentially a Counter mode with added authentication. The Atmel chip using the CCM mode does not allow to output a message without MAC and to decrypt a message without verifying MAC first.

6.1.4 Authentication and message integrity

As described above, due to the CCM mode the new system provides authentication and ensures the message integrity. In the existing system authentication is based on the fact that only the chip and the cloud share the secret key, and also on the advertisement nonce included in the encrypted message passed from the lock to the cloud and back. The cloud does not know whether the message received is coming from the chip or not – it can be, for example, an older message sent by an attacker. In the new system the attached MAC and nonce used during its computation ensure that the message is indeed created by the chip and it has not been tampered with during the data transmission.

6.1.5 Lock-smartphone data transmission channel

Both systems use Bluetooth without security for transmitting the data from the chip to a smartphone and back. No changes have been made and Bluetooth security is not proposed for use (see Section 3.3). The reason for not using this possibility is that the security of the system is based on AES-CCM encryption with the shared key only known by the chip and the cloud server. Therefore, providing additional security in the channel is not necessary for data security. However, this may lead to BLE channel authentication failure, see Section 6.2.4.

6.1.6 Smartphone-cloud data transmission channel

Both systems use protected HTTPS channel for exchanging the data between the smartphone and the cloud server. No changes were made in this part of the system. The main purpose of using HTTPS is to prevent possible impersonation of the legitimate smartphone by an attacker, and it keeps on being the same in the new system. This attack is described in detail in Section 6.2.1.

6.2 Attack resistance

This section will describe some common attacks and discuss how well the existing system and the new system resist these attacks. The common attacks we will be discussing are the Passive Eavesdropping attack, the Man-in-the-Middle (MitM) attack, the Replay attack, and some generic attacks on proximity channels.

6.2.1 Passive eavesdropping attack

The passive eavesdropping attack is very common and simple to perform. When two parties exchange messages between each other, an attacker is situated between them and observes all the messages. However, the attacker does not change existing messages or create new ones; instead she just makes use of the observed data.

In the case of the Smart Lock Access Control System, this attack might be used to steal the user credentials of the legitimate user. In this scenario, an attacker is located between a smartphone and the cloud server (see Figure 6.1).

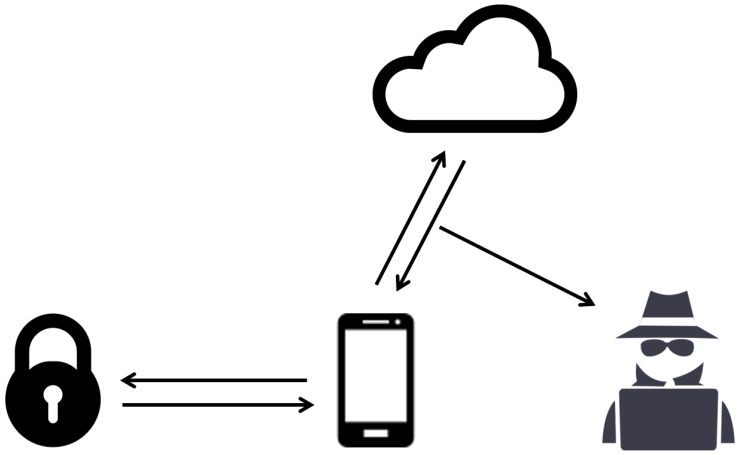

Figure 6.1: Passive eavesdropping attack: adversary between a smartphone and the cloud

The attacker aims to steal the user credentials sent from the smartphone to the cloud along with the encrypted advertisement packet in order to re-use it for herself the next time.

The countermeasure used in both the existing system and the new system for preventing such attacks is HTTPS connection. When the smartphone receives an advertisement packet from the lock, it uses secure HTTPS connection to forward the packet along with the user credentials to the cloud server. The attacker will not be able to steal the user credentials.

If the attacker is located between the lock and a smartphone and observes the messages sent over the Bluetooth channel, she will not receive any useful information either. All the packets sent there are encrypted by AES, and the attacker cannot decrypt them since the key is shared only between the cloud server and the lock.

Thus, both the existing system and the new system are secure from the eavesdropping attack.

6.2.2 Man-in-the-Middle attack

This is one of the most common attacks in communication protocols. In the attack, an attacker is situated in between two communicating parties and impersonates one or both of them. Unlike the previously discussed Passive Eavesdropping attack, in this case the attacker is active: she does not only observe the messages but can also change them or create new ones.

In this scenario an adversary is in between the lock and a smartphone. She impersonates the lock to the smartphone and vice versa (see Figure 6.2). We assume that the adversary is able to intercept and change all the messages sent from the lock to the smartphone and back.

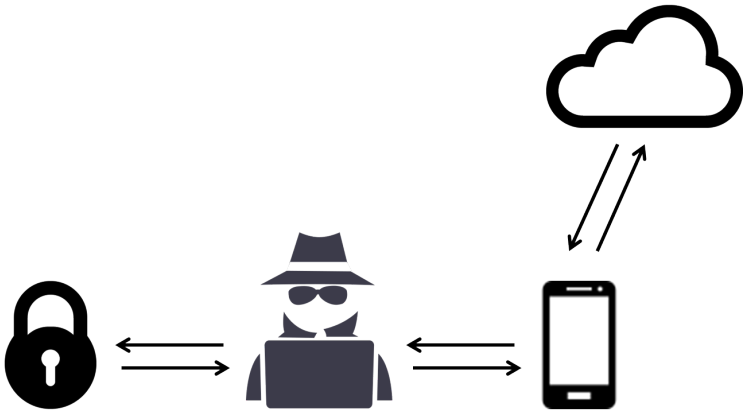

Figure 6.2: Man-in-the-Middle attack: active adversary between the lock and a smartphone

In both the existing system and the new system, an adversary cannot form an advertisement packet herself, because she needs to know the secret key. Replacing an original advertisement packet with something else will lead to decryption failure and rejection of the packet by the cloud server.

Imagine the situation when an adversary impersonates the lock for getting an opening message from the cloud. In the existing system she will receive only encrypted data, while in the new system it is received along with a MAC and a CCM nonce in clear. In both cases, this data does not help the attacker: she cannot decrypt the opening message due to lack of the secret key and the knowledge of a CCM nonce does not help her achieve anything: the nonces are different for each opening message and they do not have to be secret in principle. Replacing an original opening message with something else will lead to decryption failure and rejection of the message by the chip.

The adversary might want to corrupt a CCM nonce sent in clear in the new system. For example, she replaces it with a previously used one, in order to collect more advertisement packet ciphertexts encrypted using the same key and CCM nonce (see Section 6.2.3). The chip will fail to decrypt the opening message with the corrupted CCM nonce. The chip then will need a CCM nonce to encrypt the next advertisement packet. In this case the chip will use the previous “successful” CCM nonce stored in the User Memory.

The first attempt of the cloud server to decrypt this advertisement packet using the CCM nonce later corrupted by the adversary will fail. However, the backup CCM nonce is also stored in the cloud server database, so the server will be able to decrypt this advertisement packet from the second attempt. The cloud server then knows that the last CCM nonce was corrupted and will not save it in its database as a new backup CCM nonce.

The attacker may still collect more advertisement packet ciphertexts encrypted using the same key and CCM nonce (for example, for a future Replay attack), if she keeps on corrupting the CCM nonce. Then the chip will be forced to always fall back to the same backup nonce stored in its memory. However, in order to achieve this the attacker needs to wait for legitimate users trying to open the door a significant number of times in a row. The attacker needs to keep on collecting pairs of advertisement packets and opening messages until a collision of advertisement nonces occurs. Using a birthday paradox, we can compute that in order to achieve a collision probability of 50%, the attacker needs to collect 77164 pairs of packets. Obviously, this is not reachable in practice, because the attacker has to keep on corrupting CCM nonces: after one successful door opening a CCM nonce will be updated and the adversary will have to start collecting data from the beginning. The door not opening many times in a row (i.e., the lock not working) will cause suspicion from the legitimate users and lead to replacement or fixing of the lock.

The same result will be achieved when an adversary is located between the cloud server and a smartphone. The adversary cannot reach the data in both the existing and new systems due to the use of HTTPS channel. Corrupting an advertisement packet leads to its rejection by the cloud. Correspondingly, corrupting an opening message will result in the lock rejecting it later on. Changing a CCM nonce sent from the cloud server will be rejected by the system similarly as the previous scenario.

We can conclude that both the existing system and the new system would resist the Man-in-the-Middle attacks.

6.2.3 Replay attack

A Replay attack is based on recording the messages sent by legitimate parties to each other in order to re-use them later.

Imagine that an adversary intercepted a message sent from the lock to the cloud server, recorded it and wants to re-use it later. In the existing system the cloud server will not notice that due to the lack of authentication. The cloud server will decrypt the message, check the data inside (this will obviously be correct because it has been approved before). The server does

not store previously used advertisement nonces, so it will not notice that the advertisement nonce is being re-used. It will send the opening message back to the lock. However, the chip will not accept this message because the advertisement nonce inside the message has already expired and does not match the one currently stored in the EEPROM.

The new system will also resist this attack but at an earlier stage. Since the messages sent from the lock are encrypted using a CCM nonce generated by the cloud server on the previous step, the cloud server will not accept such message due to MAC verification and decryption failure. The adversary cannot re-encrypt the message with a current CCM nonce because she does not know the shared key needed for this.

Another possibility for the adversary is to re-use an opening message sent from the cloud server to the lock. To avoid timestamp check failure, she sends along a "correct" timestamp recorded during the legitimate unlocking operation. This attack will be resisted by both systems for the same reason: the advertisement nonce inside the encrypted message will not match the one stored in the chip memory.

As in the case of the MitM attack, an adversary might start collecting pairs of advertisement packets and respective opening messages, waiting to get a collision of both the advertisement nonce and the CCM nonce. However, this will require $2.17 \cdot 10^{19}$ records to reach the collision probability of 50%. Obviously, this is way beyond the number of openings during the Smart Lock lifetime.

Concluding, both systems successfully resist the replay attack. However, the new system does it at an earlier stage, thus saving time and resources.

6.2.4 Channel hijacking and mafia fraud

For completeness, we describe some generic attacks which apply to most systems that use proximity to provide security.

The first such attack is hijacking the Bluetooth channel. An attacker stops an opening message sent from a smartphone to the lock and re-uses it later when the legitimate user has gone away. The system cannot resist such an attack. The recommendations for preventing it might include initial instructing of the users to not go away from the door if it does not open but contact administrators of the system instead. Also, another means of physical security may be used, such as guards or security cameras.

Another type of attack is an insider attack, which means that among the legitimate users there is one who is willing to help an attacker to open the lock.

In such a scenario, the legitimate user can be far away and is able to prove that his phone was not in the proximity of the lock at the time when the fraudulent opening of the lock happened. The legitimate user provides his user credentials, including the secret information, to the attacker. The attacker uses this to convince the cloud server that she has access rights to unlock the smart lock. The cloud server will send an opening message and the door will be unlocked. The logs of this operation will be stored in the cloud server database. However, later the legitimate user will be able to complain, claiming that he was not present at the scene when the door was opened.

Neither the existing system nor the new system provide a defence strategy for this kind of attack. Moreover, resisting such attacks is out of scope of this thesis. One recommendation to avoid possible lawsuits to the manufacturer might be to include a statement to the license that the manufacturer is not responsible for fraudulent openings when an attacker gets access to the user's smartphone or the user credentials.

Chapter 7

Overview of the Smart Lock Market

A few other commercial smart lock solutions are available on the present market and will be discussed in this chapter. Unfortunately, most of the manufacturers do not divulge exact information about the cryptography behind their products. We know that usually the locks use AES to encrypt communication between a lock and a user, but exact schemes of authentication are mostly not disclosed.

Apart from the general descriptions, some weak points will be listed, along with examples of successful attacks on the smart locks.

Products of four big companies will be presented in more detail. These smart locks are August, Kwikset Kevo, Goji, and Lockitron. A few other products will be mentioned briefly.

7.1 August smart lock

The August smart lock is one of the most popular commercial products of this type available on the market. It is easy for physical installation: it does not replace the whole lock but only replaces the thumbturn on the interior of the door (see Figure 7.1). In order to use it, one needs to download and register an August smartphone application. This application allows an owner to issue both permanent keys and temporary keys (for example, for guests). The system still allows using a normal key to open the door [6].

The same three parties are involved in the system: a smart lock, a user with a smartphone and a cloud server. The lock communicates with the smartphone over BLE in Just Works mode, and the smartphone is connected to the server over HTTPS. The August system uses 128-bit key AES in CBC

Figure 7.1: August smart lock

mode to provide secure communication between a lock and a smartphone. To establish a session, both the lock and the phone generate a 64-bit long random number. The procedure is described in Figure 7.2 [3] [11]. Once an encrypted session is established commands can be issued to the lock.

The main Printed Circuit Board (PCB) of the smart lock includes an ultra low-power STM32L152 microcontroller, BLE SoC (System on a Chip) CC2541: the integrated circuit combines microprocessor core and other peripherals, in this case a Bluetooth transmitter [27], Motor Driver DRV8833, MCP6142 Operational Amplifier, and Power Amplifier LM4861M for the speaker. Apart from Bluetooth communication, the CC2541 chip is responsible for encryption and decryption of packets and key management. The phone connects with the chip over BLE, and then it sends a request to open the lock to the STM32L152 microcontroller [17].

Each lock can store up to 256 AES encryption keys. The key in slot 0 is the most privileged key: only a session established with this key can fill in any of the other key slots. This key is shared between the lock and the August cloud server and it appears to be impossible to change. All the keys are stored in the CC2541 chip's flash memory that causes a serious breach in the system security described in [2]. The design of the August system allowed all legitimate users to read the flash memory of the chip, including the key 0. This would allow any user, including a guest, to gain access to the lock indefinitely.

This attack, presented at the big hacker conference DEF CON in July

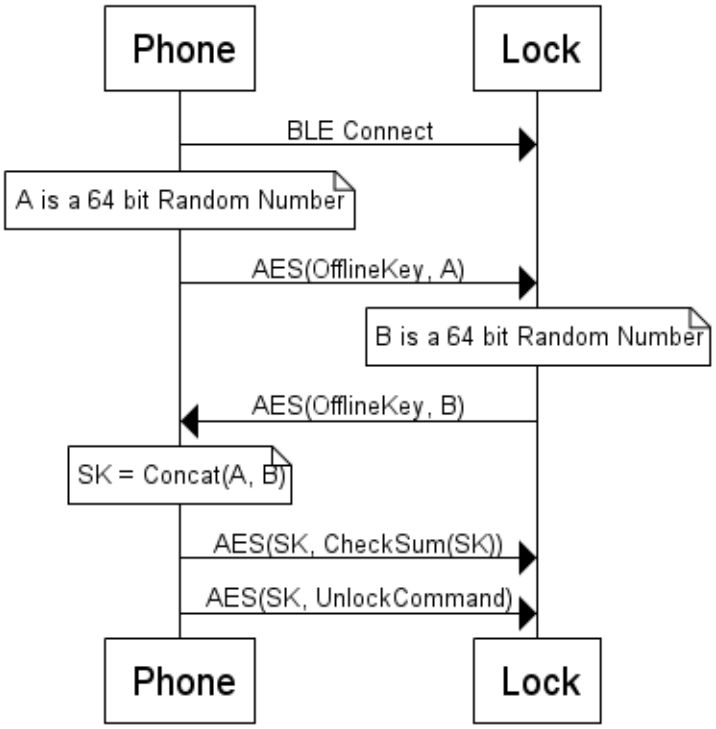

Figure 7.2: August offline session establishment scheme

2016 [11], summarised the main mistakes made by the developers of the system:

- Mobile application logs include key material
- Lock does not differentiate between guest and owner
- Firmware not signed
- No apparent way to discover backdoor keys
- Guest users can download key material
- Access entry log can be erased by guest users
- Confusing two factor authentication with two step authentication
- No rate limiting of password reset attempts
- Mobile applications include bypass for certificate pinning

- SecureRandom not used for nonce or session key generation
- Key material not stored on iOS keychain

It was noted though that the following features prevent possible attacks:

- Mobile applications attempt to use certificate pinning
- Protocol makes use of CBC nonces
- Not reliant solely on BLE Just Works security model

The August representatives reacted on the presentation and soon released the patches covering some of the flaws.

7.2 Kwikset Kevo smart lock

Unlike the August, the Kwikset Kevo smart lock replaces the whole deadbolt. The lock also hosts a Kwikset Smartkey, a mechanical cylinder that uses a normal key (see Figure 7.3). The system works as follows: a user has to carry a key fob or a smartphone with the Kevo application installed. When the user wants to open the lock, he just needs to touch it [19].

Figure 7.3: Kwikset Kevo smart lock

An owner can distribute three types of electronic keys (eKeys) between users:

- Anytime eKey: this key will work constantly.

- Guest eKey: gives 24-hour access rights.
- Scheduled eKey: gives access rights for certain pre-set dates and times.

As well as August, the Kwikset Kevo system does not rely on BLE's security protocol. It uses a Public-Key Infrastructure (PKI) system to authenticate users [1]. The company claims that each communication session between the lock and the user's smartphone is a unique transaction, which excludes a possibility of Replay attacks. However, the technical details of the implementation have not been published.

7.3 Goji smart lock

As Figure 7.4 illustrates, the Goji smart lock replaces the whole deadbolt. On the exterior side of the door the lock features a digital display and an integrated camera. The display greets the user by name when opens, and the camera takes a photo of the user to send it to the owner of the lock. In order to unlock the smart lock, the users have to have a key fob or an application from Goji on their smartphones [16].

Figure 7.4: Goji smart lock

The Goji smart lock has Wi-Fi connectivity integrated with BLE, as well as Near-field Communication (NFC). The Wi-Fi option allows the owner to command the lock remotely. However, this significantly reduces the lifetime of a battery.

7.4 Lockitron

Unlike the previous three systems, the Lockitron does not require any hardware replacement. It fits over the thumbturn on the existing deadbolt, as shown in Figure 7.5. The smart lock can be unlocked using a smartphone with an application, remotely via a web application, or even with a text message from a usual cellphone. There is a feature called Sense. It will sense a smartphone getting in close proximity of the lock and unlock automatically.

Figure 7.5: Lockitron smart lock

As well as the Goji system, Lockitron makes use of BLE, Wi-Fi and NFC. Lockitron works with Electric Imp on the Wi-Fi side and uses industry standard TLS to secure device connections. Sensitive data is encrypted during storage and transmission. All the traffic sent by the parties in the Lockitron system is redirected to HTTPS. Web applications accessed by the public are hardened against XSS, code injections, MitM attacks, and Replay attacks [20].

7.5 Successful attacks on different smart locks

The conference DEF CON in July 2016, mentioned earlier, also featured another presentation about smart locks. Anthony Rose and Ben Ramsey from Merculite Security tested 16 different doorlocks and padlocks based on BLE [12]. They showed that on 12 of them security was very weak or had serious flaws.

Four of the tested smart locks, viz. Quicklock Doorlock, Quicklock Padlock, iBluLock Padlock, and Plantraco PhantomLock, sent plaintext passwords in clear. Anyone passively eavesdropping on the communication between these locks and the users could get the key. In the case of Quicklock smart locks Rose could even change the admin password, effectively locking out a legitimate user.

Another four locks, viz. Ceomate Bluetooth Smartlock, Elecycle Smart Padlock, Vians Bluetooth Smart Doorlock, and Lagute Sciener Smart Doorlock, were prone to Replay attacks. In some of these systems encryption was used, but they were not protected against such kind of attacks.

Okidokey Smart Doorlock was broken by changing one byte in its encrypted key to 0x00. The lock entered an error state and opened.

Another security flaw noted by the researchers was predictable nonce. An attacker plays a classic MitM attack, sitting between the lock and a user (see Figure 7.6). The attacker connects to the lock to receive a nonce. Then she tries sending a wrong password, rejected by the lock, and contacts the lock again. The lock replies with a new nonce. Now the attacker knows how the lock forms the nonces and can predict the next one. She uses a legitimate user to pass the predicted nonce to the cloud server and receive a valid reply. Then the attacker contacts the lock again and receives the expected nonce, on which she already has a valid response. This attack broke the security of Mesh Motion Bitlock Padlock.

The team could not breach the security of the following four locks: Noke Padlock, Masterlock Padlock, Kwikset Kevo Doorlock, and August Doorlock. All these systems have the following features:

- Proper AES encryption
- Truly random 8-16 bytes nonces
- Two-step authentication
- No hardcoded passwords
- Long passwords (16-20 characters) allowed

Another possible attack on Goji and Lockitron smart lock systems is described in [1]. These smart locks are using Wi-Fi. If the Wi-Fi network goes down, the smart locks cannot download the latest list of revoked keys from the server. Then a user with a revoked key can still open the lock.

A similar attack can be played against the August and the Kwikset Kevo systems. They are not connected to the cloud directly. Then if the owner

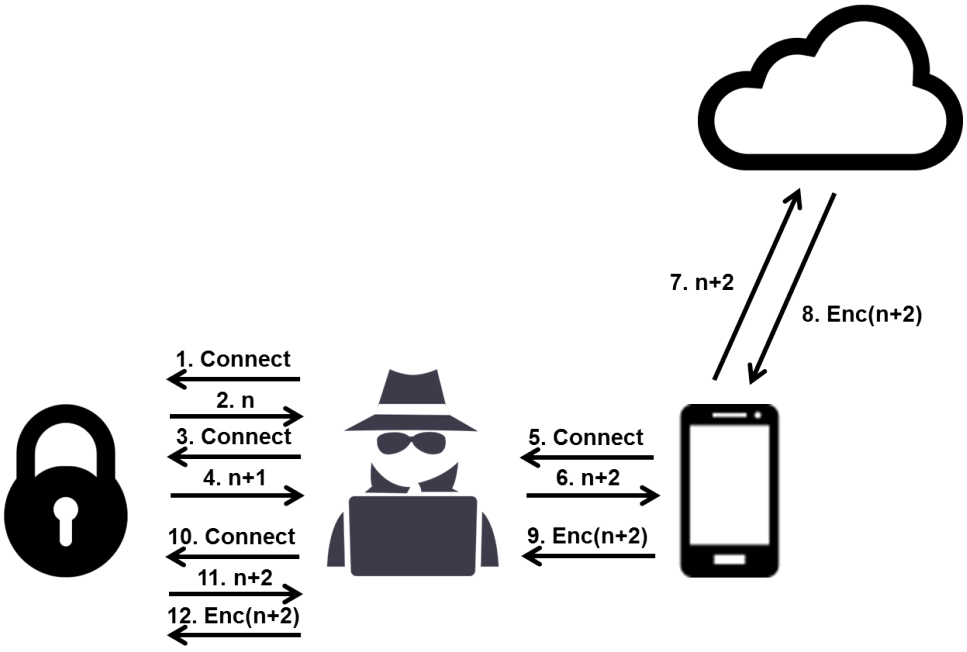

Figure 7.6: MitM attack on the smart lock system

revokes some key via web or the application on the smartphone, the lock will be updated about the revoke only after interacting with the said key. So, once the user with the revoked key connects to the lock with the application open, their key is automatically updated and revoked. However, the user can still get in if he turns off the connection to the server (for example, by putting a smartphone to Airplane mode) while Bluetooth is still running.

Chapter 8

Security Advantages of the New System

Due to very little information being available about the actual functionality of the competing smart lock systems described in the previous chapter, it is hard to compare them with the new designed system in detail. But this chapter will go through the attacks mentioned in Chapter 7 and discuss the weak spots on an example of the new system.

We start by discussing simple attacks. As mentioned in Section 7.5, Quicklock Doorlock, Quicklock Padlock, iBluLock Padlock, and Plantraco PhantomLock are susceptible to a passive eavesdropping attack. As discussed in Section 6.2.1, the new system is protected from eavesdropping, as data sent between the parties is encrypted by AES and protected by using an HTTPS channel. The only data sent in clear are the CCM nonce (along with an opening message from the cloud to the lock), system data in part of the advertisement packet, and a timestamp. The CCM nonce does not have to be secret, and the timestamp is not secret information either. The system data from the advertisement packet (described in detail in Section 3.2.1) does not have any secret value and will not help to decrypt any message or weaken the system security.

Another attack described in the same section broke the security of the Okidokey Smart Doorlock. The researchers changed one byte in its encrypted key to 0x00, which caused the smart lock to enter the error state and open. This is an example of a MitM attack. As discussed in Section 6.2.2, this or similar attack will not cause the same response in the new system. Changing bytes in the encrypted messages will cause a decryption failure and lead to rejection of the packet by the lock or the cloud server.

The Ceomate Bluetooth Smartlock, Elecycle Smart Padlock, Vians Bluetooth Smart Doorlock, and Lagute Sciener Smart Doorlock all turned out to

be vulnerable to Replay attacks. This kind of attack was discussed in detail in Section 6.2.3. It was concluded that the new system is secure against Replay attacks.

Predictable nonces were a vulnerability in the Mesh Motion Bitlock Padlock. The attacker could predict the next nonce and use a legitimate user to get a valid response on it from the cloud server (see Figure 7.6). Both the CCM nonces and advertisement nonces in the new system are generated randomly for each new packet, so they are impossible to predict. It is also worth mentioning that even if the CCM nonces were predictable (for example, the cloud server would just increase a value of the previous nonce by one, instead of generating a new random number), this would not cause a breach in security of the system. The advertisement nonce encrypted in the message would not match and the lock would reject the packet.

As stated in Section 7.1 the August smart lock allowed any user to read all the keys, including the most privileged key in slot 0. The keys in the August system were stored in the flash memory of the CC2541 chip. This is not possible in the new system, because the ATAES1232A chip does not allow anyone to read any data from the Key Memory.

Generally speaking, a strong advantage of the system described in this work, over most of the other smart locks on the market, is that a user and the lock do not share any secrets with each other. The user does not play any role except forwarding messages between the chip and the cloud and adding his credentials and a timestamp to the messages. For example, once the owner of the lock revokes the key of the user and it is changed in the cloud server database, the user is no more able to open the lock, in the way described in Section 7.5. In that example, the user with the revoked key could get in by turning the Wi-Fi network down or putting his phone in Airplane mode. This will not work in the new system, because the user must interact with the cloud in order to unlock the smart lock. The chip does not store a list of valid and revoked passwords in its memory. In principle, there are no passwords that the user has to provide to the lock, he only uses some secret information to establish an HTTPS channel with the cloud server.

We can conclude that the system described in this work would resist the attacks that were successful on the other smart locks on the market. The security of the new system is not weaker but often stronger than that of the other available smart locks.

Chapter 9

Conclusions

In Section 1.2 the objectives and goals set before the start of the project were described. Let us see if these goals have been reached in the work.

The existing system has been analysed from the security point of view. Chapter 3 gives a description of the system and its security analysis, indicating the weaknesses.

The Atmel EEPROM chip has been investigated in Chapter 4. This part of the work presents the possibilities the chip offers for improving the security of the system and introduces the details used in the new system.

The main result of this work is the proposed new design for the Smart Lock Access Control System described in detail in Chapter 5. Two alternative implementation options arose during the work process. In the chip-based solution the chip generates the one-time number needed for the confidentiality and integrity algorithm, while in the cloud-based solution the cloud is responsible for this task. They have been thoroughly described and compared in Section 5.2. As the result of this analysis, the cloud-based solution has been recommended as more efficient and secure.

The system has been evaluated in Chapter 6 by comparing it to the existing system and has been examined against the other similar products on the market in Chapter 8.

As follows from the comparison, the new system is more secure than the existing one. The main advantages are as follows. First, the Atmel chip is better protected at hardware and software level. Second, the new system design provides authentication and message integrity. As a consequence, the new system will resist a Replay attack at an earlier stage and guarantees much smaller probability of a collision.

Finally, it has been shown that the new system outperforms many existing smart locks on the market from the security perspective. Chapter 7 showed that many of the other products are susceptible to simple attacks that the

proposed new smart lock system will easily resist.

The proposed system is not perfect and there is room for improvement. Nevertheless, the work reported in this Master's thesis is publicly available and can serve as a basis for the development of next generation smart locks. Possible improvements might include applying another EEPROM chip in the system, using more secure user authentication over the HTTPS channel, and adopting the security features of Bluetooth LE for protecting the short range communication between the smart phone and the lock.

Bibliography

[1] Are Smart Locks Secure, or Just Dumb?, May 2013. http://gizmodo.com/are-smart-locks-secure-or-just-dumb-511093690.

[2] August Lock Firmware Keys, Aug 2016. https://jmaxxz.com/blog/?p=550.

[3] BLE Session, Aug 2016. https://github.com/jmaxxz/keymaker/wiki/BLE-Session.

[4] Atmel Corporation. From the Sensing Edge Node to the Cloud - Security, 2015. http://downloads.codico.com/misc/JKO/Codico%20IoT%20Seminars/ATMEL%20From%20the%20Sensing%20Edge%20Node%20to%20the%20Cloud%20-%20Security.pdf.

[5] Atmel Corporation. ATAES132A 32K AES Serial EEPROM Specification, Oct 2016. http://www.atmel.com/Images/Atmel-8914-CryptoAuth-ATAES132A-Datasheet.pdf.

[6] August, Retrieved on 12.12.2016. http://august.com/products/august-smart-lock/.

[7] Bluetooth Special Interest Group. Simple Pairing Whitepaper, Aug 2006. http://mclean-linsky.net/joel/cv/Simple%20Pairing_WP_V10r00.pdf.

[8] Bluetooth Special Interest Group. Specification of the Bluetooth System. Core Package Version 4.2, Dec 2014. https://www.bluetooth.com/specifications/adopted-specifications.

[9] Cherdantseva, Y., and Hilton, J. A Reference Model of Information Assurance Security, Sep 2013. http://users.cs.cf.ac.uk/Y.V.Cherdantseva/RMIAS.pdf.

[10] Daemen, J., and Rijmen, V. AES Submission Document on Rijndael. Version 2, Sep 1999. http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael.pdf.

[11] DEF CON 24. Backdooring the Frontdoor. Hacking a "Perfectly Secure" Smart Lock, Aug 2016. https://media.defcon.org/DEF%20CON%2024/DEF%20CON%2024%20presentations/DEFCON-24-Jmaxxz-Backdooring-the-Frontdoor-UPDATED.pdf.

[12] DEF CON 24. Picking Bluetooth Low Energy Locks, Jul 2016. https://media.defcon.org/DEF%20CON%2024/DEF%20CON%2024%20presentations/DEFCON-24-Rose-Ramsey-Picking-Bluetooth-Low-Energy-Locks.pdf.

[13] Evans, D. The Internet of Things. How the Next Evolution of the Internet Is Changing Everything, Apr 2011. http://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf.

[14] Ferguson, N., Schneier, B., and Kohno, T. *Cryptography Engineering. Design Principles and Practical Applications*. John Wiley and Sons, Mar 2010. page 71.

[15] FIPS Publication 197. Advanced Encryption Standard (AES), Nov 2001. http://csrc.nist.gov/groups/ST/toolkit/block_ciphers.html.

[16] Goji, Retrieved on 12.12.2016. https://www.indiegogo.com/projects/goji-smart-lock#/.

[17] Goverdovsky, A. August Smart Lock Teardown, Retrieved on 12.12.2016. https://medium.com/ruki-founder-s-journal/august-teardown-6274a7858338#.xgaadlbtq.

[18] Joint Test Action Group. 1149.1-2013 - IEEE Standard for Test Access Port and Boundary-Scan Architecture, 2013. http://standards.ieee.org/findstds/standard/1149.1-2013.html.

[19] Kwikset, Retrieved on 12.12.2016. http://www.kwikset.com/kevo/.

[20] Lockitron, Retrieved on 12.12.2016. https://lockitron.com/security.

[21] Microchip Technology Inc. 512 Kbit SPI Bus Serial EEPROM, 2011. http://ww1.microchip.com/downloads/en/DeviceDoc/22237C.pdf.

[22] Network Working Group. HTTP over TLS, May 2000. https://tools.ietf.org/html/rfc2818.

[23] NIST Special Publication 800-27A. Engineering Principles for Information Technology Security, Jun 2004. http://csrc.nist.gov/publications/nistpubs/800-27A/SP800-27-RevA.pdf.

[24] NIST Special Publication 800-38A. Recommendation for Block Cipher Modes of Operation: Methods and Techniques, Dec 2001. http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf.

[25] NIST Special Publication 800-38C. Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality, May 2004. http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38c.pdf.

[26] NIST Special Publication 800-90A. Recommendation for Random Number Generation Using Deterministic Random Bit Generators, Jun 2015. http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf.

[27] Texas Instruments. CC2541 2.4-GHz Bluetooth Low Energy and Proprietary System-on-Chip, Jun 2013. http://www.ti.com/lit/ds/symlink/cc2541.pdf.

[28] Ulusoy, C. Android Library Design and Implementation for Smart Lock Access Control Systems. Master's thesis, Department of Communications and Networking, Aalto University School of Electrical Engineering, Espoo, Finland, 2015. http://urn.fi/URN:NBN:fi:aalto-201512165695.

Appendix A

ATAES132A Chip Security Configurations

The tables presented in this Appendix describe all possible configuration settings in the ATAES132A chip. These settings are applied for the User Zone, Key, VolatileKey, and Counter registers, mentioned in Section 4.2.3.

A.1 User Zone configuration

Each User Zone Configuration Register has the following configuration settings:

ZoneConfig Field	**Description**
AuthRead	Requiring authentication to read data.
AuthWrite	Requiring authentication to write data.
EncRead	Requiring encryption to read data.
EncWrite	Requiring encryption to write data.
WriteMode	Defining if the zone is read/write or read-only.
UseSerial	Requiring SerialNum to be used in EncWrite operations.
UseSmall	Requiring the first four bytes of the Small Zone to be used in EncWrite operations.
ReadID	Setting KeyID to be used to encrypt the data read from this zone and generate the MAC.
AuthID	Setting KeyID to be used for inbound authentication before access is permitted.
VolatileTransferOK	Permitting key transfer from this user zone to VolatileKey.

ZoneConfig Field	Description
WriteID	Setting KeyID to be used to decrypt data written to this zone and to verify the MAC.
ReadOnly	Defining if the zone is read/write or read-only. Can be updated after locking the Configuration Memory using the `Lock` command.

Table A.1: User Zone configurations in ATAES132A

A.2 Key configuration

Each Key Configuration Register has the following configuration settings:

KeyConfig Field	Description
ExternalCrypto	Permitting to use `Encrypt` and `Decrypt` commands with the key.
InboundAuth	Permitting to use only the `Auth` command for Inbound Only or Mutual Authentication with the key. Other commands' usage is prohibited.
RandomNonce	Requiring a random Nonce to be used with the commands using the key.
LegacyOK	Permitting to use the `Legacy` command, i.e., to use AES in ECB mode.
AuthKey	Requiring prior authentication using the KeyID stored in LinkPointer.
Child	Permitting to use the `KeyCreate` or `KeyLoad` command with the key.
Parent	Permitting the key to be used as the parent when writing VolatileKey via `KeyCreate`, `KeyImport`, or `KeyLoad`.
ChangeKeys	Permitting the key updates by the `EncWrite` command after locking the Key Memory.
CounterLimit	Enabling the limits for usage count for the key.
ChildMac	Requiring an InMAC to modify the key using the `KeyCreate` command.
AuthOut	Enabling I^2C Auth signalling for the key.
AuthOutHold	Defining if the I^2C AuthO output state is reset or unchanged when an authentication reset is executed using the key.

KeyConfig Field	Description
ImportOK	Permitting to use the `KeyImport` command with the key.
ChildAuth	Requiring prior authentication for the `KeyCreate` command using the KeyID stored in LinkPointer.
TransferOK	Permitting to use the `KeyTransfer` command with the key.
AuthCompute	Permitting to use the `AuthCompute` command with the key.
LinkPointer	Setting KeyID to be used as the Parent Key (for child keys) or as the authorising key.
CounterNum	Storing the CntID of the Monotonic Counter attached to the key for usage limits or for MAC calculation.
DecRead	Permitting to use the `DecRead` and `WriteCompute` command with the key.

Table A.2: Key configurations ATAES132A

A.3 VolatileKey configuration

There is also a seventeenth key register, named VolatileKey. It is used as a temporary session key, which is stored in SRAM. VolatileKey also has a Configuration Register. This register has the following configuration settings:

VolConfig Field	Description
AuthOK	Permitting to use the `Auth` command with the key.
EncryptOK	Permitting to use the `Encrypt` command with the key and requiring prior authentication.
DecryptOK	Permitting to use the `Decrypt` command with the key.
RandomNonce	Requiring a random Nonce to be used with the commands using the key.
AuthCompute	Permitting to use the `AuthCompute` command with the key.
LegacyOK	Permitting to use the `Legacy` command with the key, i.e., to use AES in ECB mode.
WriteCompute	Permitting to use the `WriteCompute` command with the key.

VolConfig Field	Description
DecRead	Permitting to use the `DecRead` command with the key.

Table A.3: Volatile Key configurations in ATAES132A

A.4 Counter registers configuration

The CounterConfig registers put restrictions on using Counters. Each Counter can increment up to 2,097,134 using the `Counter` command, and they cannot be changed afterwards, neither they can decrement. The Counters are used to prevent exhaustive key search attacks. The CounterConfig register has the following configuration settings:

CountConfig Field	Description
IncrementOK	Permitting to use the `Counter` command for increments with the key.
RequireMAC	Requiring an input MAC for increment operations.
IncrID	Setting KeyID to be used to generate an input MAC for `Counter` command increment operations.
MacID	Setting KeyID to be used to generate an output MAC for textttCounter command Counter read operations.

Table A.4: Counter register configurations in ATAES132A

Appendix B

ATAES132A Chip Commands

As noted in Section 4.5, the ATAES132A chip has 24 specific cryptographic commands, apart from standard EEPROM `Write` and `Read`. Table B.1 [5] gives a short description of every command.

Command	Description
`Auth`	Performs one-way or mutual authentication using the specified key.
`AuthCheck`	Checks the output MAC generated by the Auth command or by reading a counter using the Counter command on a second ATAES132A device.
`AuthCompute`	Computes the input MAC required to execute the Auth command or to increment a counter using the Counter command on a second ATAES132A device.
`BlockRead`	Reads 1 to 32 bytes of data from User Memory or the Configuration Memory. Returns cleartext data.
`Counter`	Increments a high endurance Counter and/or returns the counter value.
`Crunch`	Processes a seed value through the internal crunch engine. This function is used to detect clones.
`DecRead`	Checks the output MAC and decrypts data that was encrypted by the EncRead command.
`Decrypt`	Decrypts 16 or 32 bytes of data provided by the Host after verifying the integrity MAC.
`EncRead`	Encrypts 1 to 32 bytes of data from User Memory and returns the encrypted data and integrity MAC.
`Encrypt`	Encrypts 16 or 32 bytes of plaintext data provided by the Host.

Command	Description
`EncWrite`	Writes 1 to 32 bytes of encrypted data into the User Memory or Key Memory after verifying the integrity MAC.
`Info`	Returns device information: MacCount, Authentication status, or hardware revision code.
`KeyCreate`	Generates a random number, stores it in Key Memory, and returns the encrypted key to the host.
`KeyImport`	Decrypts and writes a key that was output by the KeyCreate command.
`KeyLoad`	Writes an encrypted key to Key Memory after verifying the integrity MAC.
`KeyTransfer`	Transfers a key from User Memory into the Key Memory or into the VolatileKey Register.
`Legacy`	Performs a single AES-ECB mode operation on 16 bytes of data provided by the Host.
`Lock`	Permanently locks the Configuration Memory or Key Memory. Locked memory can never be unlocked.
`Nonce`	Generates a 128-bit Nonce from the internal RNG for use by the cryptographic commands. This command can also be used to write a Host Nonce directly into the Nonce Register.
`NonceCompute`	Generates a Nonce in a manner that allows two ATAES132A devices to have identical Nonce values.
`Random`	Returns a 128-bit random number from the internal RNG.
`Reset`	Resets the device, clearing the cryptographic status.
`Sleep`	Places the device in the Sleep state or Standby state to reduce power consumption.
`WriteCompute`	Encrypts data and generates the input MAC required to execute the EncWrite command.

Table B.1: Commands embedded in ATAES132A